Game Design Tools : Can They Improve Game Design Practice?

Katharine Neil

Conservatoire National des Arts et Métiers

Ecole Doctorale Informatique, Télécommunications et Electronique

Groupe ILJ - Equipe MIM - Laboratoire CEDRIC

THESE présentée par :

Katharine NEIL

soutenue le : **18 Décembre 2015**

pour obtenir le grade de :

Docteur du Conservatoire National des Arts et Métiers

Discipline/ Spécialité : **Informatique / Multimédia**

Game Design Tools: Can They Improve Game Design Practice?

These dirigee par:

NATKIN Stéphane — *Professeur, CEDRIC, Cnam, Paris*
SWALWELL Melanie — *Professeur (Associate Professor), Flinders University, Australie*
DE VRIES Denise — *Maitre de conférences (Senior Lecturer), Flinders University, Australie*

Rapporteurs:

ESTRAILLIER Pascal — *Professeur, Université de La Rochelle*
CHAMPION Erik — *Professeur, Curtin University, Australie*

Examinateur:

LE PRADO Cécile — *Maitre de conférences, CEDRIC, Cnam, Paris*

ACKNOWLEDGEMENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I certify that this thesis does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

ABSTRACT

This thesis contributes practice-led evaluation research to the question of whether game design tools can effectively support and expand game design practice. It offers insights that can be used to inform future game design tool development.

Game designers, unlike most other design practitioners, typically do not use design tools in their work. While conceptual models and software tools have been developed to address this, we lack discussion and critique of how tools work in practice.

Taking the point of view of a practitioner, this study of game design tools is based on longitudinal, practice-led evaluation research conducted as a participant-observer, applying game design tools to 5 contrasting game design case studies.

Design tools support game design practice. In Chapter 2 I set out what that practice is today, reviewing the game design process and design activities, and giving particular attention to contemporary trends and directions.

Chapter 3 situates game design within the broader context of Design Studies. After reviewing relevant ideas from this literature, I conclude that game design is best characterised as a crafts-based design discipline, and that adoption of design support would mean a shift towards "self-conscious" design. I argue that, within this context, the adoption of tools into current practice would represent a significant and disruptive change to current practice. I then trace the history of development of game design tools within the game design and research communities.

Chapter 4 offers a more precise definition of game design tools and presents a comparative review of the tools within this scope.

In Chapter 5 I argue for the need to bring practice-led evaluation research to this question. Here I set out my research goals and questions, before specifying the particular tools I practiced with for this study. I discuss methods used in relevant research fields and outlines the method used for this project.

Chapter 6 introduces the game projects I used as case studies and describes how I worked with each. Chapters 7 and 8 present observations and analyses of my experiences using a selection of

tools that vary in their approaches. These include (though not exclusively): *Articy:Draft* (Nevigo GMBH 2011), *Machinations* (Dormans 2009), *Ludoscope* (Dormans and Leijnen 2013), and "progression planning" (Butler et al. 2013). Evaluation themes and criteria are drawn from the Design Support and Creativity Support Tool literature. Chapter 9 refocuses attention on the practicalities, addressing the question of how game design tools integrate into the wider context of practice.

In Chapter 10 I present and discuss my experiences with my own game design tool, which I prototyped in order to evaluate and extend one of the tool approaches under study. I explain my tool design choices, some of which reflect the knowledge I have acquired through the course of this study.

Chapter 11 summarises my conclusions in relation to how design tools might best support game designers, and offers ideas for further practice-led evaluation research related to this question.

Keywords:

game design, design tools, practice-led, videogames, evaluation research

RESUME

Cette thèse constitue une recherche expérimentale pour évaluer la capacité des outils d'aide à la conception du « Game Design » de jeux vidéo. Elle cherche à dégager les éléments destinés à instrumenter et à améliorer le développement de ces outils.

La production des jeux vidéos s'appuie sur de nombreux outils techniques allant des éditeurs graphiques et sonores jusqu'au moteurs de jeux. Mais, en amont, et à l'inverse des praticiens d'autres domaines du design, les concepteurs de jeux (Game designers) utilisent généralement peu d'outils dans leur travail quotidien. Bien que des modèles conceptuels et des outils logiciels aient été développés pour les assister, il manque une étude approfondie sur la pratique de ces outils. En prenant le point de vue d'un « Game Designer », notre étude se base sur une étude de terrain, menée par un participant–observateur, utilisant les outils sur 5 études de cas.

Nous analysons dans un premier temps les pratiques du Game Design : les différentes activités impliquées et leur mise en œuvre dans le processus de conception. Nous nous intéressons plus particulièrement aux développements et tendances actuelles.

Le chapitre 3 positionne le Game Design dans le contexte plus général des travaux en cours sur la critique et l'analyse du Design (Design Studies). Nous en concluons que le Game Design se situe plus comme dans le domaine de la conception artisanale (« craft-based ») et que l'utilisation d'outils serait une évolution vers une conception réflexive (« self-conscious »). Cette évolution constitue donc, à notre avis, une rupture par rapport aux pratiques actuelles. Nous retraçons l'histoire du développement des outils de Game Design tant du point de vue de la recherche que de l'industrie. Ceci nous permet, au chapitre 4, de donner une définition plus précise et comparative des outils de Game Design.

Le chapitre 5 formalise les questions de recherche de cette thèse et la nécessité d'une évaluation basée sur la pratique. Le point de vue méthodologique de notre travail est ensuite développé.

Nous présentons ensuite nos observations et analyses d'expériences réalisées en utilisant une sélection d'outils reposant sur différentes approches. Parmi ces outils citons: « Articy:Draft », « Ludoscope » et « Progression Planning ». Nos critères d'évaluations sont tirés des publications sur le « Design Support » et le « Creativity Support Tool ».

Au chapitre 10 nous décrivons et discutons les expériences effectuées avec un outil de design que nous avons développé dans le but d'évaluer et d'étendre l'approche « progression planning ». Nous explicitons les choix de conception de l'outil issus en partie du travail d'analyse mené dans cette thèse.

La conclusion résume nos observations et introduit des principes d'amélioration des outils de « Game Design » et propose d'étendre nos travaux sur différents types d'outils et contextes d'utilisation non abordés ici.

Mots clés :

jeu vidéo, game design, conception, outil de game design, étude de terrain

1.   INTRODUCTION

The art form of the video game has a very idiosyncratic reliance on the process and practice of its designers; they work with creative and computational problems that form a web of deep complexity. This is both a blessing and a curse to game designers: the real-time algorithmic and interactive dynamics of play are often too complex to be modelled or evaluated successfully on paper, or even in the mind of a designer. While designers in other creative fields – with their graphical notation systems (music), story-boarding (film), sketching and modelling (architecture) – have meaningful ways with which to abstract, externalise, analyse and evaluate their ideas before taking them into production, game designers currently do not. Game designers, unlike most other design practitioners, typically do not use tools to design.

Dating back more than a decade, key game industry figures and academic researchers have sporadically but consistently identified our lack of design tools as both a problem and an opportunity. They have argued that formal, abstract tools for designing gameplay (be they conceptual models or software) must be developed if the craft of video game design is to attain desired levels of sophistication and creative expression.

Towards this goal, theoretical work has been undertaken to formalise and abstract game design techniques into formal models, to create taxonomies and to develop graphical notation systems (Björk and Holopainen 2005; Koster 2005; Araújo and Roque 2009; Natkin and Vega 2004; Bura 2006; Sicart 2008; Reyno and Cubel 2009; Cook 2007). More recently there have been a few attempts by researchers to concretise some of these ideas into software tools (Dormans 2009; A. M. Smith, Nelson, and Mateas 2010; Karakaya et al. 2009).

The practitioner community, however, remains sceptical and has largely ignored this work, leaving the question of whether design tools can meaningfully support and expand the practice of game design unanswered. Even within the research community there has been relatively little discussion or comparative critique of a kind that can inform future advances in this emerging field.

To enable any such a discussion we need practical evaluation of proposed design support approaches that goes beyond simple research validation. This doctoral research project aims to contribute evaluation research towards this end. Taking the point of view of a practitioner, this

study of game design tools is based on longitudinal, practice-led evaluation research conducted as a participant-observer, applying game design tools to longitudinal game design case studies.

I have looked specifically at videogame design, though many of the tools for videogame design would be useful to tabletop or pen-and-paper games as well. In addition, due to the scope of this work as a doctoral research project, my case study projects have had a design and development team of one person (with one exception, where I collaborate with a visual artist on game narrative for one of the case studies). As such, I have had to infer what design experiences with tools might be like within larger design teams and more typical development contexts. To imagine this I have drawn on my background as a game designer, programmer and sound designer within the game industry in Australia and France, ongoing freelance work in small-team based mobile game development, and participation in industry workshops and conferences. My perspective is also coloured, as the reader may notice in the way I approach design tools, by an earlier background in musical composition, the path to my doctoral studies being via an undergraduate education within the discipline of Music.

Game design is many things to many people, depending on their viewpoint and agenda. I have met people in the arts and entertainment world who are most comfortable understanding game designers as creatives who write stories that happen to be interactive and require computers. I have also met people with a technical background who imagine game designers as user experience designers who write technical specifications for the development of software, the function of which happens to be to entertain the users. Neither of these perspectives is strictly untrue - they simply demonstrate how widely variable our range of disciplinary understandings can be.

To analyse my experiences with these tools and my case studies, therefore, I look to a range of research disciplines. First, the work that has been done directly on the topic of game design tools within the disciplines of Game Studies and Computer Science and Human Computer Interaction. In particular, however, I draw upon knowledge from domains where the discussion on tools and their relationship to design practice is more advanced: Design Studies, Design Support Tool and Creativity Support Tool research. In order to understand some of the domain-specific design challenges faced by game designers, I also consider aspects of game design culture and practice, as well as contextual factors such as game industry trends that influence game design today.

I have sought to address the following research questions:

1. Can game design tools improve game design practice?
2. Can game design tools expand the scope of game design outcomes?
3. How do design tools change or impact the game design process?

More generally, I hoped to discover knowledge that could contribute to future game design tool development research.

1.1. Structure of this thesis

Design tools support game design practice. In Chapter 2 I set out what that practice is today, reviewing the game design process and design activities, and giving particular attention to contemporary trends and directions**.**

In Chapter 3 I look more broadly at design practice. For this I draw theories from Design Research – a domain that some games researchers have identified as a promising potential framework for understanding game design experiences. Following a brief introduction to Design Research, I present some of its concepts and discuss their relevance to game design process. Domain-specific differences and challenges are also covered. From within this context I conclude that game design is best characterised as a crafts-based, vernacular design discipline, and that adoption of design support means a shift towards "self-conscious" design.

Chapter 4 traces the history of the calls for and the development of game design tools, a research domain within both academia and industry that aims to give game design the support it needs to allow it to develop into a mature design practice.

Chapter 5 offers a definition of game design tools that I use for the purpose of this study, then a review and comparison of the tools the research community and industry have produced that fall within this scope.

In Chapter 6 I argue for the need bring practice-led evaluation research to this question. Here I set out my research goals and questions, before specifying the particular tools I practiced with for this study.

Chapter 7 discusses methods used in relevant research fields and outlines the method used for this project. As a participant-observer/"reflective practitioner" conducting practice-led research, I describe how I apply an understanding of the design process to observe and analyse my experiences with the game design tools under study.

Chapter 8 introduces the game projects I used as case studies and describes how I worked with each.

Chapters 9, 10, and 11 elaborate my observations along with analyses. This content is structured around and framed by evaluation themes and criteria drawn from the Design Support and Creativity Support Tool literature. While Chapters 9 and 10 focus on the tools themselves and the act of using them, Chapter 11 widens the focus to using the tools in the context of practice.

In Chapter 12 I present and discuss my experiences with my own game design tool, which I prototyped in order to evaluate and extend one of the tool approaches under study. I explain my tool design choices, some of which reflect the knowledge I have acquired through the course of this study.

Chapter 13 concludes this thesis with a summary of findings and some potential directions for future work.

2. THE PROCESS AND ACTIVITIES OF GAME DESIGN

2.1. Overview

Design tools support game design practice. In this chapter I review what that practice is today.

First, I give an overview of the literature on game design activities and process - of which there is surprisingly little. I then review game design tasks - highlighting the rise in importance of progression design, followed by a review of game design process and design activities. I give particular attention to contemporary trends and directions, which include data-driven design and "game jam culture**".**

2.2. A paucity of literature

Any analysis of the effect using design tools has on game design process requires a reference point: an understanding of current game design process. Here we come upon a problem, however. As I will argue in later chapters, one of the reasons for the lack of progress in tackling the question of design support in the game design community is the lack of formalisation: designers have not yet codified their practices.

One might expect, however, a systematic study of game design practice to have been carried out, if not within the industry, at least within the academic sphere. Such material could provide a useful reference and starting point for my study. This is largely not the case, however. Kuittinen and Holopainen, who analysed six game design books to see how the literature in our domain corresponds to the theories of the general design research, found that even the most popular texts ostensibly written to teach game design (e.g. (Fullerton 2008; Salen and Zimmerman 2005)) stop short of describing the game design activity itself, focusing instead on game analysis (Kuittinen and Holopainen 2009). Holopainen and Nummenmaa add that game design as a design activity is treated by the literature as 'monolithic', a mysterious 'black box' (Holopainen, Nummenmaa, and Kuittinen 2010). Where design activity is implied, it is brainstorming:

> There is a tendency towards equalling solution generation to brainstorming game ideas, which are then gradually revised into game designs through an iterative process (Fullerton 2008; Rouse and Ogden 2005; Salen and Zimmerman 2005; Schell 2008). This view is somewhat problematic because it hides the intricacies of solution generation under the heading of brainstorming thus making it harder to understand and talk about the mechanisms behind it. It also suggests that

solutions are only created at the initial stages of the process thus further blurring the idea that design takes place throughout the whole development cycle (Kuittinen and Holopainen 2009).

And according to Järvinen (Järvinen 2008, 48):

> …in my experience most of the literature functions at its best on an inspirational level (e.g. Koster 2005), or is strongly design-orientated (Salen & Zimmerman 2004; Fullerton et al. 2004). These are important contributions as such, but they rely quite a lot on the reader's personal ability and experience to find practices and methods to transform the inspiration into concrete results.

Indeed, the games research community has paid little attention to game design process (perhaps the exception being study of the process of making serious games). Sociologist Jennifer Whitson points to the lack of developer-centric research in general within Game Studies, and attributes this to issues of access, the game industry being, until recently at least, a notoriously insular community with developers typically being legally bound by non-disclosure agreements. I suspect it is also a disciplinary issue: the fact that Game Studies is situated mainly around disciplines that focus on, and have the developed tools to analyse, audience and technology (Literature, Media Studies, Computer Science, Human Computer Interaction (HCI), etc.), rather than design practice and practitioners. The exception is the growing research focus on design practice being initiated by Artificial Intelligence (AI) researchers who are working in the areas of game design support, procedural content generation and automated design (often within the field of Computational Creativity). It has been predominantly these kinds of researchers – ones whose job it is to create solutions – who have expressed an interest in breaking the 'black box' of game design down into definable processes and activities that can be formalised, and, in the cases of procedural content generation and automated design, for example, reproduced[1].

Some of these researchers have found it useful to look to Design Studies for guidance and I have done this also in the chapter that follows. In this chapter, however, I have been compelled to supplement academic research with industry publications, as well as knowledge I have gleaned as an industry participant.

---

[1] This scenario - where computer scientists find themselves tasked with trying to make up for a lack of foundational design research, and end up providing some of the groundwork of design theory themselves - is not unprecedented. Design research itself has historically been instigated by interdisciplinary researchers with a background in artificial intelligence (Herbert Simon, for example).

2.3. Game design tasks today

Here I look at what game design is today, from the point of view of the tasks of game design.

The term 'game design' is generally understood to mean the crafting of the core player experience ('gameplay' or 'rules'), rather than the visual, narrative or software components of a game. A person employed in the role of game designer is responsible for the design of a game artefact as opposed to its construction. While some designers may have production tasks, where they are responsible for producing or editing game data (often mission editing or statistics for game balancing), the primary role of a game designer is that of conception and communication throughout the course of the project. They are responsible for generating, developing and problem-solving "gameplay" – i.e. the player experience.

In the 1990s and 2000s, as team sizes increased and games became bigger productions, design roles were split into specialisations: Game design and level or mission design. Narrative design and technical design too, in larger teams. Typically, in large scale game development today, game design roles are broken down into specialist roles. The main specialisations are game design, level design and narrative design/writing. They vary according to genre, but these roles are typical in AAA[2] games.

"Game design", if used to designate a sub-category of game design (as distinct from the sub-category "level design", explained below), typically means performing design tasks concerned with the mechanics, rules or system elements of the game, and how the player interacts with that system (what Chris Crawford described as the game's "verbs" (Crawford 2003, 166)). A game designer is usually involved in the tasks of ideation and concept development in the early stages of design. Later tasks involve designing the "rules" and broad system aspects of the game: objectives, dynamics of the game etc. system for a first-person shooter, or figure out jump heights and special

---

[2] In this thesis I am using a definition where AAA signifies medium to large scale, multi-million dollar budgeted game productions, as distinct from "indie", "social" (i.e. Facebook), and mobile games.

attacks for a platformer. "Technical" or "systems" design is a specialisation within the game design role, tasked with the design of game sub-systems and game economy[3].

Level design takes those elements and creates content for the player to experience over time – within a space (i.e. game level) or within a narrative comprising missions, for example. While game design is conventionally the systems-oriented, number-crunching task; level designers, though also responsible for balancing player skill and difficulty in their work, conventionally more concerned with world and story and moment-to-moment gameplay, and less number crunching.

Among the game writer roles, the narrative designer designs the narrative for the game, in collaboration with the design team. Level/mission designers may write mission dialogue themselves, or this may be assigned to writers.

These roles are often broken down even further in large studios. For example, a designer might be responsible for spawn-point placements within the levels – and that remains their sole design task until they are promoted to a different game design job. There are also exceptions to the practice of assigning specific design roles: for example, Valve's "Cabal" system specifically does not separate out design from production roles (Birdwell 2006), and small independent development teams often see artists and programmers performing design tasks.

2.3.1. New design tasks emerge

The tasks of game design within the industry are changing. Independent game development has allowed game design to explore new directions, from which new genre currents have emerged. These include design for procedurally generated gameplay, sandbox games experiences (e.g. *Minecraft*), and "not games" (games without defined objectives, e.g. *Dear Esther*). However, as I describe below, arguably the most significant changes to the practice of a game designer have come along with new platforms and business models.

---

[3] According to Rollings and Adams: "An economy is a system in which resources move around, either physically from place to place, or conceptually from owner to owner. [...] Most games contain an economy of some sort. Even a first person shooter has a simple economy: Ammunition is obtained by finding it or taking it from dead opponents, and it is consumed by firing your weapons" (Rollings and Adams 2003, 234)

2.3.1.1. The rise and dominance of progression design

Jesper Juul has given us a handy distinction between "games of emergence" and "games of progression":

> The history of computer games can be seen as the product of two basic game structures, that of emergence (a number of simple rules combining to form interesting variation) and that of progression (separate challenges presented serially). Emergence is the primordial game structure, where a game is specified as a small number of rules that combine and yield large numbers of game variations, which the players then design strategies for dealing with. This is found in card and board games and in most action and all strategy games. Emergence games tend to be replayable and tend to foster tournaments and strategy guides. Progression is the historically newer structure that entered the computer game through the adventure genre. In progression games, the player has to perform a predefined set of actions in order to complete the game (Juul 2002).

Juul tells us that all pre-electronic games are games of emergence; simple rules leading to complex gameplay, while progression-based games are "historically new".

I would argue that in the domain of contemporary digital games, especially in the AAA space, games with strong progression structures dominate. While the rise of more experimental independent productions has offered a wider range of game genres, this structure is still dominant in that space as well. This is reflected in the breakdown of roles in larger design teams, where there are often many level designers compared to the number of game designers.

The nature of game progression has been changing, however. So too have the kinds of design tasks designers find themselves having to perform. Significant changes to the nature of design work over the last decade reflect the rapid ascendance of new game genres, platforms and business models.

One of the most disruptive trends to affect game design practice has been the rise of the games-as-service model. Games-as-service, now the dominant model worldwide for mobile games and moving into the core games space (LeJacq 2012) arose initially in East Asia, where freemium payment models emerged (for both casual and core games) largely in response to piracy. Key to this model is long term retention of players, as games-as-service is typically monetized using a "freemium" payment model: players pay progressively via micro-transactions as they play rather than as an up-front purchase.

Game design for the games-as-service paradigm has brought some interesting challenges to the distinction between games of emergence and progression. While Juul argues that most games

contain combine elements of both emergence and progression, games-as-service has provided a new incentives to find ways of introducing and superimposing progression structures upon game genres that are, in terms of their core gameplay, games of emergence. *Candy Crush*, for example, offers quick, match-3 gameplay, but it is the player's long-term relationship with the game – broadly speaking, their progression – that is key to the game's success. Sandbox style management simulation games such as *Rollercoaster Tycoon* are now played within the context of heavily engineered progression re-emerging as games like *Farmville*; puzzle games are now predominantly packaged as levels with elements progressively introduced.

In many such games the progression is heavily engineered. Progression can, in a sense, become the core gameplay mechanic itself, where the player's decision-making and meaningful play is wholly contained within the game's progression design. For example, the moment-to-moment activity may be as simple as clicking or tapping a button (as satirised by Ian Bogost's Facebook game *Cow Clicker*) while the interest and strategy of the game is that of returning to the game repeatedly and strategically to engage with the game's progression elements. A game's progression can become so complex over such a long period that the progression itself (the acquisition of skills and powers and items) can have emergent properties of its own[4].

Even the conventional "premium"[5] console and PC game experience has been influenced: meta-game elements in the form of points and badges are now awarded for all console games. Even large-scale productions have begun adopting this trend to some extent, as AAA games begin to adopt the free-to-play business model (notably PC games such as *DOTA2*)).

Hence progression design is increasingly an important and complex design task. Later in this chapter I cover how this challenge is being met by the game industry in terms of changes to game design practice.

---

[4] *Pokémon* games are a good example of this.

[5] A "premium" pricing model has the player purchasing the game outright i.e. as opposed to "freemium" (or "free-to-play") model where the game is monetized via in-game advertising or purchases.

2.4.   Game design process today

There is little in the literature to guide us on what the game design process actually is, as part of, but distinct from, the game development process.

According to Kuittinen and Holopainen's 2009 survey of the game design literature almost all texts describe how game design is split into different stages or phases. All models described are variations on a theme: design, test, and analyse – and one is linear while the other is iterative. One set of literature uses what Kuittinen et al describe as a "stage model" where the design passes linearly from one stage to the next e.g. initial idea, concepting, designing, prototyping, implementing, playtesting (Kuittinen and Holopainen 2009). Others advocate a model within which design emerges through rapid evolution and iteration of concrete prototypes ranging from simple paper ones to complex, and almost finished, software implementations (Kuittinen and Holopainen 2009). This iterative model is newer and widely considered a "best practice" ideal – though not always possible within business constraints and large scale productions.

> The game design process is often [15, 8] described as proceeding in an iterative spiral where the basic activities of the designers keep repeating until a satisfactory solution is reached. Another popular way [1, 8, 2, 13] of describing the process is to view it in terms of succeeding stages, usually described as concept design, pre-production, production, and post- production. Whereas these models can be used to describe the process itself, they appear to be hardly descriptive of design as an activity (Holopainen, Nummenmaa, and Kuittinen 2010).

Kuittinen and Holopainen note that this design process model contains, somewhat recursively, "design" – which is not broken down into different types of actions and activities so as to enable us to analyse or discuss what the designer actually does when they develop a concept into a design, or how they set about modifying their design after duly playtesting a prototype that has been built for this purpose (Kuittinen and Holopainen 2009).

I would argue that this reluctance or inability to describe design is symptomatic of the fact that game design has been, to date, very much dependent on and somewhat dominated by, game production: game production is now, more than ever, treated as a game design activity. Over the course of this chapter I will show how and why.

2.5. Game design activities

2.5.1. Conventional design activities

The description below reflects design as it is conducted in the context of a conventional game development cycle within medium to large development teams. Even development that has adopted some "agile"-inspired[6] iterative processes (for example sprints, documentation using user stories, etc.) often still broadly follows these conventions.

Typically (i.e. in a standard commercial game development context) game designers perform design tasks using a combination of natural-language-based documentation followed by (or concurrent with) prototyping a playable version of selected elements of their design.

Documentation is considered to be a game designer's primary task and manifestation of anything that could be called a design "method"[7]. Until recently, it has been the convention to author a game design document (a 'GDD') which can run to hundreds of pages in length. These large documents, which were often demanded by game publishers, are difficult to maintain and read and have a tendency to quickly become out of date. Indeed, modern game design books de-emphasise the role of GDDs (Schell 2008, 54:54; Fullerton 2008, 446). A more modern approach is to produce short, disposable design documents that target specific areas of the design, written as and when they are needed, to be replaced rather than necessarily updated. Documentation is created using word processing, spreadsheet and data flow (e.g. *Microsoft Visio*) software packages. Diagramming is most often used to map out certain details about user interfaces, narrative flow, maps, screen wireframes and statistics. While there are no standard models or visual vocabulary to express core gameplay concepts, individual designers sometimes create their own ways of diagramming or sketching their ideas visually 'on the fly' (Salen and Zimmerman 2003), often improvising different ways to express their ideas for different projects. Predominantly, however, the game mechanics, (the 'rules') are expressed in natural language.

---

[6] Agile development. See (Beck et al. 2001)
[7] Possibly the earliest attempt to define a game design method is the game design document (Kreimeier 2003).

Following documentation, a software prototype of the design is usually created by production staff (programmers, artists) under the direction of designers, who as designers typically lack the production skills required to effectively produce the prototype themselves.

Prototyping is considered a critical game design tool, in that it makes up for the perceived weaknesses of game design documentation. Producing a playable version of the design – a prototype – is considered to be the only reliable means of evaluating and describing the emergent behaviour of produced game designs. Koster observes that "building a game off of a game design document is like trying to film a movie off of the director's commentary"; Salen and Zimmerman assert that important questions such as "is the game accomplishing its design goals?" and "are [players] having fun?" can never be answered through conception and design alone - they can only be answered by play (Salen and Zimmerman 2003).

When creating a prototype, designers must decide which aspects of the design they want to test. Usually this is a core game mechanic, but it could also include any game mechanics that are novel. In this way, the prototype is targeted; it is not simply a smaller, rougher version of the game. Several prototypes may be produced, each dedicated to evaluating different ideas. Designers are usually not the only clients of the prototypes; the evaluation of key art and technical challenges of the game could be included in the same prototype.

Designers aim to learn from the prototype and integrate their conclusions into the design. Within a conventional game development process (now falling out of favour), the prototype is "thrown away" after preproduction and the game is built from scratch, the designers being unable to playtest a version of the game again until "alpha" – i.e. towards the end of development. Iterative project management styles, facilitated by advances in game development technology, have more recently afforded designers the ability to playtest throughout the project.

Organised user testing, often conducted by specialists in user research, can provide playtesting feedback to designers that serve as a resource to support design thinking. Even quality assurance testers - whose function is primarily to find bugs – can provide designers with useful data. Designers can provide quality assurance testers specific test cases, e.g. "how many hours does it take to complete every mission in the game", or "can every boss fight be won using this character class".

The gaming public has also been used as a playtesting resource by designers. Public and private beta testing for some core game genres – particularly Massively Multiplayer Online games – has been practised for many years. Where the game is played online, it has been possible to gather data about the player experience via the internet. Offline games have relied on cooperation of the player to go online to send these data to the developer in the form of written feedback or log files.

2.5.2. Modern directions in design activities

2.5.2.1. Design by making

More recently, prototyping has become accessible to designers. A designer-accessible prototyping method that has been made possible (within the independent and amateur sectors of the game industry in particular) is the use of simplified production tools, some of which have removed the need for the user to know any form of programming to create a simple game.

This has meant prototyping has become a mainstay of design practice. It has also spawned a phenomenon within the culture of game design known as the "game jam", where designers rapidly design and produce a game or game prototype within the space of a day or two.

Another prototyping – or perhaps better called "simulating" - method is paper and physical prototyping, inspired by the methods of board game designers. It usually involves using cheap materials like cardboard and plastic tokens as abstract representations of game elements to play out a 'real-world' prototype of the game where humans perform the role of the computer as well as the players. This 'analogue' style prototyping is encouraged as best practise by the major game design texts (Fullerton 2008; Salen and Zimmerman 2003). Publisher and developer Electronic Arts gives its internal designers workshops on physical prototyping methods (Fullerton, 2008: 20).

2.5.2.2. Design "house style"

Some studios have developed their own particular design conventions and practices. It is difficult to find material on these, or to know how much exists, as internal processes are rarely shared openly. That said, designers frequently switch between projects and employers, and based on my personal associations with designers working around the industry, it is hard to get the impression that any major studios are hiding secret practices that are significantly different or much more elaborate than others.

We do know, however, that Ubisoft, for example, call aspects of their design practice "Rational Design". According to an intern who wrote about the methods he was taught at Ubisoft, Rational Design is "a tool for facilitating solid learning and difficulty curves and even helps us to inject a lot of variety into the game experience" (McEntee 2012). Within the context of progression design it appears to be the measuring and distribution of gameplay elements in terms of difficulty and variety. Design thinking is supported with simple tables and charts, for which the data are derived from playtesting and is extended by deductive reasoning and simple calculations.

In addition, over the years designers and teams have occasionally shared their own design methods in trade magazines (*Game Developer* magazine and on its associated website *Gamasutra*). Level design approaches, diagramming methods and the use of *Microsoft Excel* to calculate game balancing data are popular themes.

2.5.3. Testing

Though testing has always been part of the game development, in recent years testing has become more extensively and intensively applied to the design process.

2.5.3.1. Automated testing

As described above, designers receive input from teams of quality assurance testers. Real players (the designer, the team, the Quality Assurance department – and depending on the nature of the project, public beta testers) play a version of the game feeding that back into the game design in the form of written reports. With the games as service model (mentioned above), where games last weeks and months, internal playtesting is harder. As discussed above, progression design has become technically challenging. Conventionally, progression design has been tested by quality assurance testers. But this is no longer enough.

Automated quality assurance testing is now used to supplement manual, human methods. Automated testing has become common practice in the software industry and has now become adopted in game development. As well as verifying gameplay logic (for example: (Marlin 2011)), it can be used to produce gameplay traces that can answer simple range-based questions a designer could have that could be scripted. Rather than using deductive logic, an automated testing script can, by simple brute force, "play" the game many times over to verify game data. An automated testing script for a free-to-play game, for instance, might check for changes in the game's "pinch point" (the point at which the player runs out of game currency and will need to make a purchase

with real money in order to continue playing) after each change the designer makes to the game data. I have created such a script.

Automated testing, however, has difficulty testing the player experience. The creation of automated players is in fact a current research challenge (Nelson 2011). A script can exhaustively play a game but it cannot make gameplay choices as a player. This is a gap that can be filled by player metrics, which I discuss below.

2.5.3.2. Players as testers

Above I mentioned that in the games-as-service model an evolving relationship between the player and the game over time is a designed element of the gameplay itself. Typically, Games-as Service design process is very similar to the "lean start-up" method (Ries 2011). A Minimal Viable Product (MVP) (i.e. a deliberately small version that is not the final product), a term borrowed from the software development industry, is built expressly for design and testing purposes. The MVP is "soft-launched" in a single market (i.e. in one country) – it is not a product launch so much as a public beta test. Over a period of weeks, player metrics are gathered in real-time, analysed by game designers and specialist analysts, and used to inform modifications to the game's design. If a certain part of the game is little used by players, for example, the designers may conclude that this part is not successful and it may be cut; if players fail too many times at a task that task may be simplified. A/B testing – borrowed from Interaction Design – is also used, both for balancing gameplay and making decisions about the visual and interactive elements of the game. Once the game has attained certain behavioural goals, visible in the metrics (typically relating to player retention and monetization behaviour), the game design is considered strong enough to launch globally – this time as a product. While the game is live in the global market, small design changes are made and new content is added, but it is in the soft-launch phase that major design changes to the "core game loops" are meant to take place.

In this way design thinking is supplemented by the capture and analysis of player data, which are relied upon to iteratively reshape a game's design, particularly its progression. These new game design activities involving the preparation for, the gathering of and the analysis of player metrics – often called "data-driven design" – have become dominant in the activities of game design in this new section of the industry. Design roles dedicated to this are becoming more prevalent.

This is leading a shift in game design practice more broadly. As the games-as-service monetization model gains traction in core games, working with player metrics data is becoming a common part of the design process for more and more genres of games, on a variety of platforms (Whitson 2012, 310). Another contributing factor to the increasingly widespread use of player metrics is that it is now technically more achievable. Of course, as mentioned above, gathering of player metrics using public and private beta testing has been used for some game genres for many years – notably for online games. Now many more game genres require "always-online" play, and even single-player PC games are often played "online" – the default mode of Steam, the dominant PC gaming platform, is online. Development support for the gathering of player metrics is now far simpler as well, with the relatively recent availability of tools built into popular game engines that plug into online analytics services, many of which are game-specific. *Unity* (Unity Technologies 2005) plugs into *GameAnalytics* (GameAnalytics 2010), for example, while *GameMaker: Studio* (Yoyo Games 1999) partners with *Flurry Analytics* (Flurry 2005).

Another contemporary trend that also harnesses player data is the practice of "open alpha". This is popular in indie game development. Access to the alpha requires payment from players, and its function is primarily that of fundraising for further development on the game, and community building ahead of the game's official launch. An open alpha also provides feedback to designers. The design success of Minecraft, which ran arguably the most famous open alpha, is widely attributed to the long period of tweaking to the design based on feedback from the community during its open alpha.

In this way, testing has been extended and formalised into a powerful design tool, as part of the iterate-and-test formula now generally accepted as best practice in game design. In fact we can see that, generally speaking, the emerging trends in game design activities, from game jams to data-driven design, fit into and reinforce an iterate-and-test design narrative.

2.5.4. The influence of the software and internet industries on game design

This iterate-and-test method within game development did not emerge in isolation; it came, in large part, from trends in the software development industry – specifically iterative development models. This is not unusual – game development has long been influenced by the software development industry. The first practitioners of videogame design were programmers (Crawford 2003, 114), and a game's technical qualities continue to be (though arguably less than they were)

an important factor in a game's commercial success. In terms of power within development studios and salary, engineers still retain a lot of power in relation to designers. It could be argued that this is necessary, the technical difficulty of game development is still significant, as is the pace of technological change. This dominance, however, could explain why game developers have been influenced by the software industry more so than design disciplines in other entertainment industries in terms of design practice. As I will argue further on, we can see this influence continuing and intensifying through into the new trends in game design practice that are emerging today.

The "stage model" for game design described by Kuittinen and Holopainen (Kuittinen and Holopainen 2009) makes sense when seen alongside a conventional game production process – it is effectively the result of game production forcing its context and constraints onto game design. Here, game designers are obliged to fit their work into a sequence that includes high concept development, pre-production, and production (alpha, beta, completion), where playtesting and major design changes are only possible months into the process, shortly after the "alpha" milestone is met. The game's publisher would demand that the full design be documented and approved before production would commence.

As Waterfall-style production processes (Benington 1956) fell out of favour in the wider software industry, game developers had similar critiques of its use in game development (see for example: (McGuire 2006; Cook 2006)). It was a process in which engineering, production and business priorities tended to dominate - often at the expense of design concerns (designers having to wait months until they can playtest a first version of their design, for example). Inspired by the wider software development industry's move towards agile-inspired production methods, thinking on game development production processes shifted accordingly. I personally recall how in 2005 I was told by a very influential game design figure that the best game design advice he could give was to read The Agile Manifesto, because "everybody" in game design (note this was design, not development) was talking about it and how it might improve game design process. In other words, designers hoped that iterative development methods could not only solve the industry's

production issues[8] but also empower game designers (again, see (McGuire 2006; Cook 2006)) The second – iterative – model of design process derived by Kuittinen and Holopainen from the game design literature reflects this new game production model – i.e. game design within a development culture where "iterative development" is now considered best practice.

Using the old production model, game design was very much at the mercy of production. Iterative development has given game designers more control, but it is control afforded not by more powerful design methods or tools, but by having more control over production. In the new model, production now serves design to a greater extent: prototyping and other forms of production are formalised as explicit steps of the design process; testing for the sake of design is formalised and extended (to online player metrics). Here again, in player metrics we see the influence of current trends – and the technologies – of the software development industry and its almost ideological belief in the power of data analytics.

Indeed, key elements of the modern game design toolset (even some of the modern language of game design - the phrase "fail early", being a notable example) that support and reinforces this "iterate-and-test" method, could be in large part attributed to the influence of the software development industry. [9]

Of course, aside from the conscious adoption of modern production approaches, other factors have played a part. Advances in technology have made it significantly faster and easier to prototype games than it was a decade ago. Business factors have also influenced these design trends. For example, the gathering and analysis of player metrics is now becoming necessary for business reasons. This is in large part due to changes to the publisher-developer relationship within game industry business models: publishers and investors who would previously have offered production funding on the strength of a pitch and due diligence now are increasingly demanding that some form of the game be already produced and proved in the market (for

---

[8] In 2004 the International Game Developers' Association blamed poor project management as one of the principle factors contributing to a workplace culture of long hours, job instability, scheduling blow-outs and a 95% marketplace failure rate (IGDA 2004).

[9] It is worth nothing that, in addition, board game and table top game designers have been using iterative methods for years – another reason for videogame designers to be comfortable with a prototype-and-test method.

example, interest generated in the game via a crowd-funding campaign, player metrics from a soft-launch, or release on a single platform) before they will publish or invest. This is now common in mobile game publishers and there are signs of established AAA publishers following this trend[10] . In this way, large sections of the game business are adopting a model strikingly similar to that of start-up business culture practices of Silicon Valley – where, not so coincidentally, perhaps, the industry's main conference, the Game Developers' Conference is held annually.

Games' relationship with the wider software development industry is important to note, as its effects flow through into game design and into the day-to-day work of practitioners. Game design is being shaped by not just the technology and platforms it uses to bring games to life; it is shaped by the design practices of and the business landscape backgrounding that technology.

2.5.5. The pros and cons of iterate-and-test game design

Here I discuss the effects of these methods on design practice. In particular, do these new methods solve the gap in the design tool chain?

2.5.5.1. Design-by-making

There are undoubted benefits to designers being able to represent their designs as playable prototypes rather than in a document – it is a superior means of communicating and exploring their ideas. Unlike the blank page that a text document offers, however, there are fundamental, unavoidable limitations in what can be explored and expressed using production tools for the sake of design.

First, attempting to design sophisticated experiences by prototyping can unwittingly privilege design elements that can feasibly be implemented within the prototyping environment. This limitation could be partially attributed to the limits of the designer's own technical skill; or, more fundamentally, due to the limits imposed by the tool. This is inevitable: tools that allow designers to create rapid prototypes do this by severely simplifying, constraining and restricting the possibility space available to the designer in order to do this. This is a serious problem, given that

---

[10] See, for example, Square Enix's "Square Enix Collective" (http://collective.square-enix.com/)

design tool researchers believe that tools strongly affect a designer's thought processes (Resnick, Myers, and Nakakoji 2005)**.**

Secondly, the production goals of production tools are in direct conflict with the maintenance of the kind of unresolved ambiguities in the design situation considered effective in a design tool[11]. Production favours a direct path towards a goal, whereas design favours the exploration of a problem space. Lawson noted that for architects, production tools (in this case, 3D CAD systems) that were effective in helping architects compose solutions were found to be obstructive in terms of creative exploration (Lawson 2006).

Tellingly, improved production tools and the ease of producing a playable experience have not necessarily reduced design and development time for all genres of games. While a game can be produced over a weekend, the design work involved in the development of even relatively small games has not necessarily been reduced. It would be hard to claim that the major improvements in the design-by-making toolset have solved the wicked problem of game design.

2.5.5.2. Paper prototyping

While production tools are certainly more accessible than they used to be, they do require an investment in terms of technical skill; and, as noted above, they constrain the exploration of the designer. A paper prototype is, in one sense, an effort to participate in the prototype-and-test cycle without those constraints. It has its own drawbacks, however. Even the strongest advocates of physical prototyping advise that paper prototyping is only suitable for certain styles of games. Tyler Sigman, a designer who has made extensive use of paper prototyping in his work, observes:

> Although there are some terrific reasons to make an analog prototype of a digital game, there are also some inherent limitations to such a process. The first, and biggest, is that there are some games for which analog prototyping just doesn't make sense. Case in point, games where a real-time action component is the sole mechanic (Sigman 2005).

In a sense, one could see designers' use of production tools for design purposes, and the cheap, primitive nature of paper prototyping represent a kind of "making do" - a collection of survival strategies that designers have evolved, adapting to and around the constraints and needs of production, in the absence of investment in dedicated design technologies.

---

[11] Attributes of design tools are discussed in detail in Chapter 8

2.5.5.3. Data-driven design: "a gut-wrenching change"

With data analysis as part of a game designer's toolkit, how successful is the application of big data techniques to game design? Will big data supersede our need for design tools? If the most sophisticated tools available to designers are statistics and deriving things from data, what is the impact on design thinking?

Whitson, using a method comprising interviews with game developers and embedded ethnographies at small game development studios, discovered problematic aspects in data-driven game design.

I have argued above that the adoption of data-driven design can be seen as part of a continuation of the game industry's tendency to take its cues on development methods from the wider software development industry. As for more specific motivations for its use in games, Whitson finds that data-driven design is motivated by the need to minimise market risk:

> Metrics reduce the risk of game development. As argued by Justin Johnson, "In years gone by it could be pretty hit-and-miss, and you'd have designers that just went on gut instinct. The good ones got it right; but they didn't get it right all of the time - but now you can concentrate on looking at what all of your players want, and work towards providing it for them" (Elliott 2011b). Accordingly, rather than relying on developer insight into player behaviour, metrics are advanced as more reliable replacements for traditional modes of developer intuition (Whitson 2012, 300).

Secondly, metrics serve as a relatively cheap and easy alternative to time-consuming playtesting methods. In this, use of player metrics has been hugely successful. Here designer Laralyn McWilliams describes the benefits of using player metrics in her work on *Free Realms*:

> Metrics gave me information, but they also let players talk to me directly and honestly. This wasn't the "guess, ship, and pray" design process of console games. I wasn't making decisions in a void anymore. With a combination of understanding the game, tracking changes, watching metrics, and listening to players, we made significant improvements to the player experience (McWilliams 2013).

There are, however, many designers who are critical of the industry's use of data-driven design (including McWilliams). First, some consider it to have a negative and narrowing impact on design thinking; second, it disempowers designers rather than empowers them; it discourages risk-taking and stifles innovation; it entrenches an iterative process of evaluation that discourages innovation.

I would add that, most crucially, it is limited as design support due to its reliance on production. Like conventional user testing, the gathering of player metrics requires the production of a game to test. Just as conventional development saw costs and schedules spiral, production costs and

development time to launch an MVP suitable for gathering metrics are rapidly rising, as intense market competition raises quality expectations among players.

Both critics and advocates of data-driven design warn that these techniques should not be used as a substitute for design thinking, as their utility is strictly limited. Player metrics provide information, but analysis of that information is up to the designer. Metrics can tell the designer what their players are doing but not why; it can highlight problems but give little insight into their causes (Sinclair 2013; Fahey 2013).

Critics of data-driven design express the concern that in practice these techniques are in fact substituting for design thinking. An overreliance on analytics sees design thinking put in competition with, and subordinated to, decision-making based on player metrics. This results in games as "mathematically tailored experiences, targeting the lowest common denominator" (Bruno 2010 in Whitson, 2012:304) Whitson warns of the danger of advocates of analytics putting "too much faith in the numbers" to guide their design decisions. Whitson has observed of developers that:

> …once developers get in a metrics-driven mindset, one where minor changes are made with clear evidence to say what was right or wrong, they are limited from considering other ways of thinking. They stop considering whether the game's secret sauce is its sense of humor, the real-world experience shared by players enjoying the game with one another. It's difficult, if not impossible, to hold both of those mindsets at the same time (Sinclair 2013).

Whitson argues that not only can the use of metrics supplant design thinking, it can change it in negative ways. While metrics are excellent for optimising games, says Whitson, they "by their nature, stifle innovation". Metrics are fundamentally retrospective: they indicate what mechanics are currently popular, and can refine and perfect existing mechanics, but in doing so they contribute to repetitive and self-referential game design. Further, from a statistical point of view, a method of iterative adjustments, adaptions and improvements in fact acts against the chance of great design successes:

> One problem with this is that it drives success to the local maxima, the highest peak in the immediate vicinity. If there is a larger peak that can be reached, but only by going through a valley, reliance on metrics will keep games from reaching it (Sinclair 2013).

While testing would, on the face of it, seem to be a very good tool, too much testing can be the enemy of creativity. Whitson references Greenberg and Buxton's thoughts on how in HCI, early and frequent usability evaluation encourages developers to solve problems through iterative

refinement. This, they say, leads to "local hill climbing" (minute improvements along similar lines until an optimum state is found) at the expense of considering and developing much better ideas that lie further afield (Greenberg and Buxton 2008 in Whitson 2012). In this way, data-driven design serves to further entrench the risk-averse nature of the game industry.

Whitson argues that many developers, including those who use it, have these and similar misgivings about data-driven design. Such fears may seem exaggerated and paranoid, as one might assume that avoiding any overreliance on analytics is a straightforward case of designers using their own discretion to use player metrics wisely and appropriately. This assumption, however, overlooks the context of the power relations in the game industry that make it difficult for game designers themselves to direct their own practice.

There is a fear within the design community that methods based on player metrics which effectively means a loss of design control, where design thinking is replaced by a kind of design that becomes crowdsourced and "consumer-driven". Andrew Wilson, vice-president of *EA Sports* (i.e. speaking from a relatively powerful, establishment position within the game industry), describes the adoption of data-driven design as a "gut-wrenching change", where design power "is no longer about" the opinion of the game designer or producer now that:

> …we [EA Sports] are a consumer-driven development organization who are constantly changing based on real-time feedback on an hourly basis from the people who play our games (Wilson as cited by Brown, 2011)

Drawing on his experience designing in the games-as-service domain, Greg Costikyan is more explicit in describing this shift in power, with his bald assertion that games that rely on metrics "give suits the whip hand" – in other words, metrics wrest power away from designers and towards publishers (Whitson 2012, 303). While analytics data are ostensibly supposed to aid designers in their work, they can easily be taken out of context by publishers to be used against them in disputes over creative decisions. Notes Whitson, "numbers are convincing, whichever person in a design argument has the numbers is likely to win". She argues that this allows abuse, in that there is a power imbalance in terms of who is interpreting the data, and how these data influence the decision-making process of the interpreter (Whitson 2012, 303). Whitson goes so far as to claim that data-driven design is serving to reduce the industry's reliance on creative professionals, thus allowing publishers much more influence over design (Whitson 2012, 304). The result: "many designers are effectively designed-out of game design" (Whitson 2012, 309). Their

creative autonomy "eliminated", designers "effectively have robot overlords in the form of metrics" (Sinclair 2013).

As previously discussed, the question of creative control is important; it has long been an important theme in industry debates. As I have argued, an explicit desire for more creative control fuelled the rise of independent development; it also saw designers embrace iterative, agile development inspired methods, push for "designer-friendly" production tools and try to exert more control over a production process that they perceived as not serving the needs of design. If player metrics can be used as a means of reducing the creative control of designers then this new innovation of player metrics may not be bringing designers closer to this goal of creative empowerment.

The forces that impact creative control in the game industry – the business aspects - are relevant, but are not within the scope of this thesis. There is, however, another factor that is leaving designers at the mercy of metrics: the lack of any other meaningful design support. Yet this is not something that has been raised in the debates. NYU Game Center director Frank Lantz, in a piece critical of data-driven design, advocates in counterposition to this trend that designers instead embrace "divine inspiration"(Lantz 2015). As a design technique, however, inspiration alone cannot address the increasingly demanding challenges of game design practice. If the rise of data-driven design tells us anything it is that design support is wanted and needed. Player metrics are filling a design support vacuum: as Justin Johnson describes above, metrics were adopted as an improvement on the "hit and miss" approach of "gut instinct". While we have no other support for design thinking it is no surprise to see our practice become heavily reliant on big data and its heuristic, user research-based, brute force approach. Instead of countering the push towards data-driven with a high culture-style mystification of the design process we can do the opposite: we can demystify and codify game design, championing the kind of design thinking that affords designers useful tools and techniques – ones powerful enough to reduce our reliance on passive, testing-driven methods that are in turn reliant on production and the participation of thousands of test subjects.

3.   GAME DESIGN AS A DESIGN PRACTICE

3.1.   Overview

This chapter has two purposes. The first is to review some concepts from Design Studies that can provide a language and toolset to be used later to frame the discussion of my practical work presented in the chapters that follow.

The second is to use these notions to understand the state of game design practice, as described in the previous chapter, from the point of view of how game design tools fit in in relation to that practice. From this I conclude that contemporary trends in game design have served to entrench game design as a crafts-based design culture, masking the limitations of game design understanding and delaying a shift towards "self-conscious" design in game design practice.

The final part of this chapter describes the context and the history of calls for and the development of game design tools, which emerged into a research domain within both academia and industry.

3.2.   Why design studies?

As mentioned in the previous chapter, it has predominantly been Artificial Intelligence (AI) researchers in games who have found themselves enquiring into game design practice. Finding a lack of domain-specific literature on the process and activities of game design they have tended to look to Design Studies. There has in fact long been a link between the computational research community and Design Research. AI was one of the research areas of Herbert Simon, one of the early pioneers of Design Research. Design research in the 1960s and 70s was in part inspired by developments in AI and the cognitive sciences, where the endeavour to build intelligent computer systems focussed on the ability of such a system to solve ill-structured problems within an open context – which is somewhat comparable to designing (Dorst 2003). The field of Computational Creativity maintains a strong connection with Design Research.

Researchers in design support and automation are not the only ones to look to design research for answers, however. Holopainen et al have also looked to Design Research, and found it helpful for

analysing practice-based game design research. Design Research models have been advocated and applied by Holopainen, Kuittinen and Nummenmaa (Holopainen, Nummenmaa, and Kuittinen 2010)(Kuittinen and Holopainen 2009) to the analysis of game design process based on data gathered from participant observer practice-based research. They have published a first attempt at using design research models to analyse game design, where they study the process of designing a pervasive mobile phone game (Holopainen, Nummenmaa, and Kuittinen 2010). They report that the design research models allowed them to reveal, capture and analyse nuances of the activities, design situations, and design choices within the game design process studied that may otherwise have escaped them. Importantly, they report that models borrowed from Design Research delivered them more insights into their game design experiments than the game design literature itself (Holopainen, Nummenmaa, and Kuittinen 2010).

While the components of the game design activity have only relatively recently begun to receive attention from researchers, there is a wealth of material in other disciplines that can allow us to study game design as a design activity. As Hewett notes:

> Even though creative products can show substantial differences across several domains of human endeavour, there are reasons to believe that there are key conditions and processes that are not domain specific and that are associated with successful creative work regardless of the domain in which that activity takes place.(Hewett 2005)

Where Holopainen et al have begun to apply design research concepts to the analysis of game design scenarios, for my project I draw upon them for models to inform my evaluation criteria and analysis. I look at current game design practice through this lens in order to provide a kind of benchmark for comparison, and to develop an understanding of how design tools could, and even should, change this practice.

3.3. The character of design

Early design theorist Herbert Simon defined the act of design as the changing of an existing situation into a preferred one (Simon, 1969: 111). Within the context of interaction design, Löwgren and Stolterman add that to design is to create something new (Löwgren & Stolterman, 2004: 9), and according to Schön, a designer "makes things" – those things often being representations, plans, programs or an image to be constructed by other people (Winograd 1996).

While the above definitions of design seem fairly broad, the defining characteristic of design seems to be as much about what it is not as what it is: design problems, solutions and methods differ from those of the scientific method.

The definition of design as making something new means that while scientific inquiry seeks to describe what things are and how things work, design fields (engineering, architecture, social service fields) are concerned with what should be. According to Banathy, the salient intellectual process of science - a "conclusion-oriented" mode of inquiry - is analysis, its guiding orientation being reductionism. By contrast, the intellectual process of design – which uses a "decision-oriented" mode of inquiry - is synthesis, with an orientation towards expansionism (Banathy, 1993).

The goal of creating something new, of creating change, shapes the nature of design problems. Löwgren and Stolterman point out that while the designer's current understanding of a design situation (the current state of a design at a point in time[12]) is commonly referred to as the "problem", and her ideas on how to proceed are called "solutions", these are not problems in a mathematical or logical sense: there are no "correct answers" to a design problem and the understanding of the problem itself grows and changes during the design process. (Löwgren and Stolterman 2004).

This evolutionary process is unpredictable. According to Schön, unpredictability is a central attribute of design – not necessarily a defining one, but it is important. The unpredictability of design means that there is no direct path between the designer's intention and the outcome (Winograd 1996).

When surveying design process models, Lawson argues that process where stages of analysis (understanding the problem) and synthesis (generating solutions to the problem) are separate is

---

[12] According to Schön, a design situation includes many elements: materials, a sense of purposes and constraints as the designer sees them, and the designer's sense of the people who will eventually use the artefact resulting from the design process (Winograd 1996)

more of a scientific approach, whereas for designers, based on his study of architecture and science students, analysis is more integrated with synthesis. In Lawson's words:

> The behaviour of the architect and scientist groups seems sensible when related to the educational style of their respective courses. The architects are taught through a series of design studies and receive criticism about the solution they come up with rather than the method. They are not asked to understand problems or analyse situations. As in the real professional world the solution is everything and the process is not examined! By comparison scientists are taught theoretically. They are taught that science proceeds through a method which is made explicit and which can be replicated by others (Lawson 2006, 3rd revise:43)

3.4. Design morphologies

Design researchers have tried to describe the shape of the design process (the "morphology of design" (Asimov 1965)) by breaking it down into the types of design activity that the designer engages in, and when.

3.4.1. Two models

There have been numerous design process models proposed by design researchers, which I will survey below. Historically, however, there have been two major schools of thought of design research thinking in terms of how the design process should be described: the rational model and the action-centric model.

The first paradigm, the "rational" model, was the first model proposed. Introduced by Herbert Simon and Allen Newell in the 1970s (Simon and Newell 1972), this is a problem-solving approach that views design as a rational search process: the designer defines a "problem space" which they survey in search of a design solution. Where the problem space is too large, ill-structured and ill-defined to be described, the designer is faced with what Simon calls an "ill-structured problem". For ill-structured problems, the search process is to be applied to only part of the problem space, known as an "immediate problem space". The rational model outlines a sequence of stages that make up the design process, starting from pre-preproduction and continuing through into post-production and redesign.

The second paradigm, the constructionist "action-centric" model, emerged in reaction to this problem-solving approach. Later, Donald Schön, critiquing the rational model of mainstream design methodology (Schön 1983, cited in Dorst 2003), challenged its assumption that there is a definable design problem to start with. Simon's science of design could only be applied to well-

formed problems that had already been extracted from situations of design practice – i.e. they required design activity in order to form them. Schön criticised the lack of attention paid to the structure of design problems and its failure to link process to the problem in a concrete design situation – and that this "action-oriented" knowledge ("knowing-in-action") cannot be described within the paradigm of technical rationality. Schön admits, however, that this knowledge is hard to describe and define, and is often implicit knowledge acquired through professional practice (Dorst 2003).

3.4.2.   The difficulty of describing design process

The modern consensus, therefore, tends to regard the design process as very non-linear and a good deal harder to describe and define than first thought. The problem/solution paradigm is, modern researchers suggest, an oversimplification of the design process. Nigel Lawson, in his book *How Designers' Think*, suggests that attempts to provide design process models that map how a designer's attention moves between design activities reveal that there is in fact no firm route through the design process. He likens the designer's route to a "chaotic party game where the players dash from one room of the house to another simply in order to discover where they must go next" (Lawson 2006). Schön similarly observes this lack of linearity, and attributes the lack of direct path between the designer's intention and the outcome the fact that unpredictability itself is a central attribute of design. According to Löwgren and Stolterman, the unpredictability and uncertainty of design work renders every design process unique (Löwgren and Stolterman 2004: 9-10). The designer will "just have to put it (a design process) together yourself", according to Lawson, as a designer knowing that design is an iterative cycle of linked activities (namely analysis, synthesis and evaluation) is of little help to them in terms of guidance for how to design (Lawson 2006, 3rd revise:40).

Even the distinctions made between the activities within the process are in dispute. Central to modern thinking about design, according to Lawson, is that problems and solutions emerge together, rather than one following logically upon the other (Lawson 2006, 3rd revise:118). He points to evidence from a study where the process of inexperienced, student architects was compared to that of experienced practitioners: while design students may follow a more rigid process this is not the case for experienced designers, who tend to use a strategy of analysis through synthesis. Designers learn about the problem through attempts to create solutions, rather

than through deliberate and separate study of the problem itself (Lawson 2006, 3rd revise:44). This is because, according to Lawson, in design it is difficult to know what problems are relevant and what information will be useful until a solution is attempted (Lawson 2006, 3rd revise:56). Others agree: Koberg and Bagnall suggest that both analysis and synthesis continue throughout a project (Koberg and Bagnall 1972 cited in Dubberly 2004); Rittel and Webber (Rittel and Webber 1973 cited in Dubberly 2004), agree, saying that the information needed to understand the problem actually depends on one's idea for solving it. In this way, the activity of synthesis provides a tool for the activity of analysis: the process of attempting to solve the larger problem (synthesis) provides a means for discovering and examining the constituent problems (analysis).

Lawson even goes so far as to suggest that rather than seeing a problem and a solution as separate entities, they are better seen as two aspects of a description of the design situation (Lawson 2006, 3rd revise:296). Dorst and Cross confirm this, saying that a design solution is often a design problem until the design task is considered finished.

> In the majority of practical design situations, by the time you have produced this and found out that and made a synthesis, you realise you have forgotten to analyse something else here, and you have to go round the cycle and produce a modified synthesis, and so on. (Dorst and Cross 2001)

While the process may be blurry, it seems possible to define an overall shape or direction. Lawson describes how both problem and solution become clearer as the process goes on; Newkirk suggests that designers may begin by focusing on analysis (the problem) and gradually shift their focus to synthesis (the solution) as illustrated in Figure a (Newkirk 1981 in Dubberly 2004).
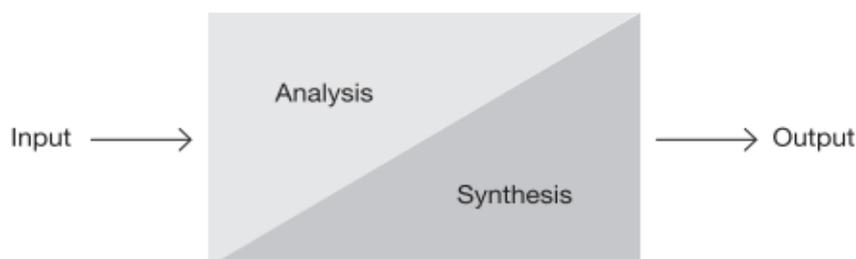
Figure a: Newkirk's design process model showing a gradual shift from analysis to synthesis (Dubberly 2004)
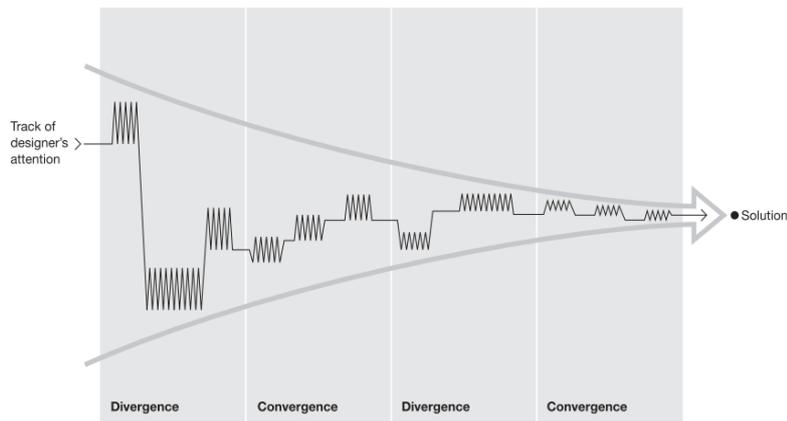
Figure b: Design morphology showing alternating periods of convergence and divergence, within a broader trend towards convergence (Dubberly 2004)

According to Nigel Cross, the overall process of design is convergent (solving), but it contains periods of deliberate divergence (exploring the problem) (Cross, 2000). This model is illustrated in Figure b.

3.5. Design activities and materials

This study is less concerned, however, with the chronological structure of the design process than with the activities contained therein, towards understanding how tools can support these activities. Even though the boundaries between design activities may be ill-defined, their descriptions provide a vocabulary with which to analyse the design experience and its artefacts.

That said, as we have seen above, design activities themselves may be hard to define, and it can be difficult to determine whether one activity is happening as distinct from another. Where distinctions are made, they can be controversial (e.g. between analysis and synthesis). Parts of the process happen simultaneously and are integrated, such that differentiating between design activities and the transitions from one activity to another pose a challenge. The activities are not, as explained earlier, separate stages in a linear process.

Below I have described elements of design activities, not from the point of view of process models, but extracted from the models and grouped according to activities. As well as surveying the design activities themselves I cover how researchers have conceptualised the materials of design (the reification of design thinking external to the designer), and how these materials are produced/externalised.

32

Below I introduce some concepts drawn from Design Studies in order to lay a foundation of underlying assumptions and terminology that I use when reflecting on practice in later chapters.

3.5.1. Design materials

3.5.1.1. Externalisation

Schön describes the act of design as the designer having "a conversation" with the materials of design (Schön 1983 in Lawson 2006). Design materials are the reification of the designer's thinking, that exist external to the designer. They represent the elements of a design situation that has been externalised from the designer's head – e.g. onto paper. Lawson calls this 'design drawing', where a drawing is not done to communicate with others but rather as part of the design process itself (Lawson 2006, 3rd revise:26). Externalising design ideas helps to lessen cognitive load (Fischer 2004). More importantly, however, by externalising the ideas the designer does not simply empty out the mind but actively reconstructs the ideas, forming new associations. As Schön notes, "it is rare that the designer has the design all in her head in advance, and then merely translates it." (Winograd 1996) In a conversation with these externalised ideas, the practitioner acts to shape the design situation by creating or modifying the materials. In return the situation "talks back" to the designer, revealing implicit, tacit, and emergent dimensions of design tasks that designers may not have considered. The designer listens and reflects on this "back talk" (Bruner 1996 in Fischer 2004). Schön describes the process as "seeing-drawing-seeing": the designer draws, and sees (and judges) what they have drawn, thereby informing further designing (Schön 1992).

Lawson characterises this externalisation as a design activity he calls "representing". A representation can be, for example, "a scribble or a functional software prototype or thinking out loud". He notes that the designer is almost always working with multiple representations – they serve as representations of possible design choices, and in this way make possible design choices concrete.

3.5.1.2. Löwgren and Stoltermans' three levels of abstraction

Löwgren and Stolterman, in their work on interaction design for information technology, distinguish three abstraction layers that describe the implicit and explicit ways design thinking is formulated at the early state of design work. According to Lawson, "representing" can be on any

of Löwgren and Stolterman's abstraction levels. They are: the vision, the operative image and the specification.

The vision can serve as a guiding, organising principle for the whole design process. It emerges (through an intuitive, rather than an analytical process) when the designer is confronted with the initial design situation. The vision can take many forms: from vague and implicit ideas to rough sketches and ad hoc verbalizations. Similar to Lawson's concept of representing multiple possible design choices, these ideas are often contradictory and chaotic, allowing the designer to be able to assess the design situation from several points of view (Kuittinen and Holopainen 2009).

The operative image, as a first externalisation of the vision, serves as a tool for making the vision and the design situation more concrete. It allows a more detailed and thorough evaluation of the design situation by allowing the designer to visualise, simulate and manipulate it (Löwgren & Stolterman, 2004: 20).

The specification describes how to construct the final artefact. Even at this stage, however, the design work is not finished as during the construction process new kinds of design situations emerge; there is no clear division between design and construction stages. A sufficiently detailed operative image can act as the specification.

All three abstraction layers form a constant, dynamic, dialectical process in which the designer moves back and forth between the layers during the design activity. The vision, which shapes the operative image and the specification is in turn shaped by them, typically becoming modified or even replaced during the later stages of the design process (Kuittinen and Holopainen 2009).

3.5.2. Analysis, synthesis and evaluation

According to Lawson, the concepts of analysis, synthesis and evaluation (also called appraisal) occur frequently in the literature on design methodology (Lawson 2006, 3rd revise:37). Lawson states that design consists of these three activities, linked in an iterative cycle.

3.5.2.1. The problem

Analysis is the ordering and structuring of the problem. Christopher Alexander described analysis as breaking ("decomposing") a problem into pieces (Alexander, 1962 cited in Dubberly, 2004). It involves the exploration of relationships, looking for patterns in the information available, and the identification and classification of objectives (Lawson 2006, 3rd revise:37). In terms of the

convergence and divergence process model described above it is the activity of analysis that creates divergence. Cross describes divergence as a widening of the search (of the solution possibility space), and to seek new ideas and starting points (Cross 2008). This decomposition-recombination process is illustrated in Figure c.
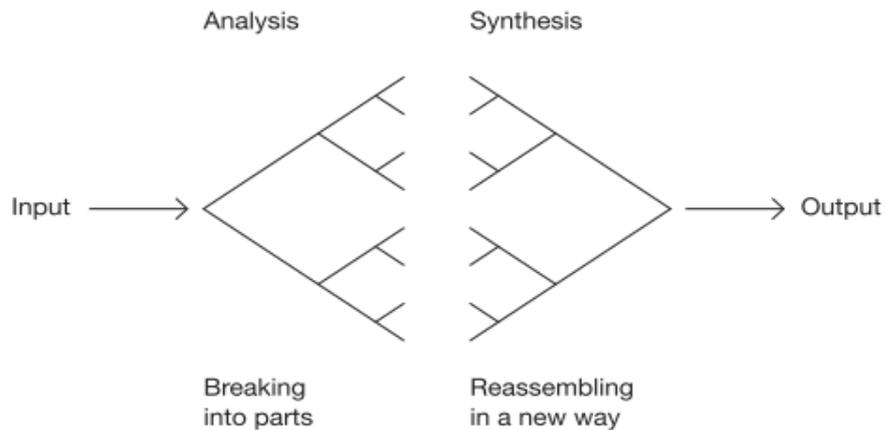
Figure c: Flow diagram showing relationships between analysis, synthesis, and divergence and convergence (Dubberly 2004)

Lawson, building on Schön's work, describes a group of activities that he calls "formulating" the design situation, by which the designer identifies the relevant elements in the situation when observing and assessing it. He makes a distinction between two types of formulating: "identifying" and "framing" (taken from (Schön 1983)).

In "identifying", the designer identifies the elements within the design situation to be able to understand their qualities and how they relate to each other. While doing this they are making judgements on the composition of the elements. Based on experience and their domain-specific knowledge, the designer recognises patterns and understands future possibilities of the situation. The representation (recalling Lawson's "representing" from above) is the medium through which the designer does the identification.

In "framing", the designer prepares the design situation for making a design move (explained below), by structuring or framing the situation as problems that the designer can attempt to solve. Framing serves as a useful tool for controlling the complexity of the design situation by allowing

the designer to focus on a select number of issues while temporarily suspending others (Kuittinen and Holopainen 2009).

3.5.2.2. The solution

Synthesis is the generation of solutions. It is characterised by an attempt to move forward and create a response to the problem (Lawson 2006, 3rd revise:37). The designer recombines the decomposed design situation. They reorder the pieces based on dependencies, solving each sub-piece, finally knitting all the pieces back together (Dubberly 2004). In the divergence-convergence paradigm, it is synthesis that creates convergence.

As discussed previously, synthesis is not dependent on analysis: researchers have observed that experienced designers do not typically wait for analysis of the problem before attempting a solution; they often develop early solutions to a design problem before understanding it.

3.5.2.2.1. Moving

When a designer alters a design situation they are "moving" – making a "design move" (Lawson 2006, 3rd revise:295). Moving comprises activities towards creating whole or partial solutions. Design moves can be "interpretive" or "lateral": that is, they can be based on reflection on the current (implicit and explicit) representations of the design situation, or they can be entirely novel or derived from existing ideas (Kuittinen and Holopainen 2009). Alternatively, design moves are "developmental": an idea is developed further and clarified, usually with some kind of representation. According to Schön, any design move has side effects, in the form of unintended and unexpected results; it is not possible to make a move that only has the consequences that the designer intends (Winograd 1996).

3.5.2.2.2. Creative leaps

A design move can sometimes take the form of a surprise, according to Schön, as the designer makes exploratory moves that allow them to see the design situation in a new way. This is commonly known as a "creative leap" (Archer 1965; Cross 1997) or what cognitive psychology would call a moment of "critical insight" (Chiang and Wang 2005) – a novel or creative solution that suddenly emerges while working on the design situation. A creative leap may follow a period of "design bottleneck": intense but vain efforts towards solving particular design problems. The designer, once having reached the limit of knowledge and heuristic reasoning, looks beyond rational thinking toward inspiration to resolve the bottleneck (Chiang and Wang 2005).

3.5.2.3. Designer evaluates their solution

"Appraisal" or "evaluation" involves the critical evaluation of suggested solutions against the objectives identified in the analysis phase (Lawson 2006, 3rd revise:37). According to Lawson, evaluation occurs throughout the design process – mostly intuitively concerning particular elements of the design, but also as judgements of the overall design situation. Evaluations can be subjective evaluations ("does this feel right") or objective evaluations that can take the form of mental simulations or user testing, for example (Kuittinen and Holopainen 2009).

Iterative cycles of evaluation seem to be a feature of the design process in many disciplines. Here in an interview about software design Schön describes how a designer asks herself evaluative questions after each design move she makes:

> As you work a problem, you are continually in the process of developing a path into it, forming new appreciations and understandings as you make new moves. The designer evaluates a move by asking a variety of questions, such as "Are the consequences desirable?" "Does the current state of the design conform to implications set up by earlier moves?" "What new problems or potentials have been created?" (Winograd 1996)

3.5.3. Meta-activities: reflecting on the design process itself

Above I have described design activities where the designer is engaged directly in the business of design. There are also activities we could call "meta-activities" – where the designer reflects on their design activities as part of the design process, or externally to the design process itself. These activities are also considered an important part of the design process.

Donald Schön has contributed the most well-known work in this area. He describes three ways in which a designer reflects on their own design thinking.

The first is "reflection in action", where the designer finds herself thinking about what she is doing in the midst of performing an action, in such a way as to influence further doing. Schön gives the example of jazz musicians improvising together, listening to each other and responding by thinking and evolving during the performance. "Reflection in action" is closely tied to the experience of surprise: when performance leads to surprise – be it pleasant or unpleasant - the designer may respond by reflection in action: i.e. thinking about what she is doing while doing it (Winograd 1996).

"Reflection on action" occurs when the designer stops to think and monitor their own process. The designer pauses to think back over what she has done in a project, "exploring the understanding that she has brought to the handling of the task" (Winograd 1996). Lawson describes reflecting on action as the designer taking a step back to look at their design process, asking themselves questions such as "are the relevant issues taken care of?", "which activities (formulating, moving etc.), if any, have been neglected?", and "am I doing this the right way?" (Kuittinen and Holopainen 2009). This kind of reflecting is important because it is the tool the designer uses to direct their path through their own design process, designing which activity to undertake and when. Asking the right question at the right time is an important design skill, according to Lawson.

"Reflection on practice" has the designer pulling back even further to reflect on their own practice more generally, based on repetitive experiences of designing. For example, says Schön, the designer may become aware of having fallen into an unfortunate pattern of design, such as "falling in love with the initial design idea", or "trying to build the diagram".

3.6.   Games as crafts-based design

Drawing on definitions from design research, here I argue that game design is currently a craft-based design practice, and that the widespread adoption of design tools would represent a shift towards conscious design. I point out that this shift has not occurred, and offer an explanation as to why.

3.6.1. Unselfconscious and self-conscious design

In comparing traditional building design to modern architecture in his 1964 book *Notes on the Synthesis of Form*, Christopher Alexander makes a distinction between two kinds of design cultures: "unselfconscious design" and "self-conscious design"[13](Alexander 1964).

"Unselfconscious design" concerns the use of traditional building methods, in which the designer works directly on the form (i.e. the building) unselfconsciously, through a complex two-directional interaction between the context and the form, in the world itself (Alexander, 1964: 77). The design rules and solutions are largely unwritten and evolve very slowly. They are learnt informally; the same form is made over and over again with no need to question why, and designers need only learn one pattern.

In self-conscious design, the design decisions are made separate to making. Form is shaped "by a conceptual picture of the context which the designer has learned and invented, on the one hand, and ideas and diagrams and drawings which stand for forms, on the other" (Alexander, 1964: 77). This is necessitated by the fact that in the modern context, new purposes and requirements frequently arise that require the development or modification of old patterns. In order that these innovations can be made, ideas about how and why things get their shape must be introduced.

Lawson describes what Alexander called "unselfconscious design" as "vernacular", or craft-based design. It is a design culture that demonstrates, according to Lawson, a "curious combination of constructional skill and theoretical ignorance". Vernacular design is practiced by artisans and craftspeople, for whom design traditions and conventions stand in place of design theory. They know that a specific design element is good, but they do not understand why. Without an understanding of underlying design principles, modifications to, or removal of, a design element can be risky. Lawson uses the example of the oddly shaped yet highly efficient design of the shape of a cart wheel, the engineering rationale for which (if it was ever understood) had long since been forgotten by the wheelwrights who fashioned them. He describes how George Sturt, when

---

[13] Alexander gives these definitions as part of an argument he makes about how architects could avoid modern American architecture's failure to successfully adapt to its context by learning from the adaptive process of unselfconscious design. For this he proposes a method he calls "mediated design" (Alexander, 1964:36).

studying the wheel's properties were surprised to discover, that the shape – hitherto believed to be a design choice born of fashions or aesthetic taste - was in fact a structural element that helps a cart balance (Sturt 1923). A lack of awareness of the reason for the wheel's design posed no problems for the exercise of craft for generations of wheelwrights. It would have, however, posed a problem for any attempt to innovate on that design.

3.6.2. Design drawing

A key factor in self-conscious design is the separation of design from making. In such a scenario, production materials no longer serve as the materials of design; the designer must purposefully create design materials – a representation of the design situation – external to the designed product itself. Lawson, in describing the emergence of self-conscious design in architecture, described how separation between design and making "urgently required" new ways of being able to model the final design, as well as support a rapid rate of change and innovation (Lawson 2006, 3rd revise:27).

These ways are known as *design drawing*:

> In a most felicitous phrase Donald Schön (1983) has described the designer as 'having a conversation with the drawing'. So central is the role of the drawing in this design process that Jones (1970) describes the whole process as 'design by drawing' (Lawson 2006, 3rd revise:26).

Design drawing is performed by the designer not for the sake of communicating with others, but rather as part of the thinking process of design itself. The design thinking becomes situated in the 'drawing' (a representation of the design situation – not necessarily 'drawing' in the literal sense), rather than the designed product. In this way, design drawing separates out design thinking from making, and in this way it is central to the self-conscious design process.

Design drawing is a powerful tool. It encourages experimentation, liberating the designer's creative imagination in a "revolutionary" way. This, according to Lawson, renders design a process almost unrecognisable to the vernacular craftsman. In effect, the designer is afforded a greater "perceptual span" that enables them to make more fundamental changes and innovations within one design than would be possible in the vernacular practice.

In terms of process, design drawing is iterative; the designer can continue the process of drawing and redrawing until all the problems the designer can see are resolved. This confers great manipulative freedom; parts of the proposed solution can be adjusted and the implications

immediately investigated without incurring the time and cost of constructing the final product. The iterative process of drawing allows designers to adapt to new design requirements as a result of changes in context – for example changes in technology that impact on design.

Design drawing can, however, be limited. In the context of architecture, while a drawing offers a reasonably reliable model of appearance it cannot necessarily model the performance or function of a structure. One might argue, however, that this could be a limitation of the design drawing/modelling tool (pen-and-paper drawing, for instance), rather than of the design drawing process itself (Lawson 2006, 3rd revise:27).

3.6.3. Game design is crafts-based design

Arguably, game designers engage in something similar to what Lawson describes as a "vernacular" or craft-based design process, or what Alexander might call "unselfconscious design". Game design best practice orthodoxy emphasises design-by-making; using artisanal, traditional knowledge, game designers are urged to prototype, they test, they make adjustments and test again.

Using Alexander's definitions, game designer Ben Cousins argued in a short blog post in 2004 that many game designers are still operating under an unselfconscious system of design.

> Many game developers show a similar inflexibility in process, and it leads to the same potential for danger. The 'high priests' and 'elders' observe the development process, and ensure the ritual takes place as they have always done it. Any deviation from the tradition is met by scorn, and most of the time the games produced show a marked similarity to previous products which have used that traditional process. Academic, scientific and analytical techniques are not allowed, as they could call into question the origins of the ritual – the search for true knowledge is withheld and the high priests and elders maintain power (Cousins 2004a).

This echoes Lawson's characterisation of crafts-based design as a "curious combination of constructional skill and theoretical ignorance". Similarly, Sicart observes, in reference to design documentation practices, that:

> most of [the game design literature that advises best-practice methods for documentation] is based on tradition or a set of common practices more than on a research-based approach to the formal elements of games (Sicart 2008).

The fact that, as noted above, "design" has featured in the standard game design literature as a step in a model of the design process itself suggests that the game design community struggles to

even distinguish game design from the game development process as a whole. In this way, design does not consciously exist independent of making.

3.6.3.1. Craft-based design in games discourages risk-taking

As stated previously, a vernacular design process, in its reliance on design conventions rather than design knowledge, has been observed to be the enemy of innovation. In relation to game design, Cousins argues that such a practice can be "damaging to the future of those designers who use it". Especially relevant to the domain of game design is one of the key challenges that self-conscious design addresses: technological change. Cousins, writing at the time as a vocal advocate for the formalisation and encoding of design knowledge, argues that we must move towards self-conscious design in order that game design meet the needs of shifting markets and advances in technologies:

> It is only by the rejection of this unselfconscious process, the constant evaluation and re-evaluation of techniques, the application of academic and scientific measurement and technique that any designer (be it a house builder, telecommunications designer or videogame developer) can make sure they don't become as much a historical footnote as the straw house builders of Polynesia (Cousins 2004a).

Cousins' observation should be seen in historical context: a time just prior to the rise of independent game development, when many in the industry were frustrated with the conservatism and lack of design innovation within their practice and sought to pinpoint causes and devise solutions. Published on the internet in 2000, *The Scratchware Manifesto* exemplifies this mood (Anonymous 2015). Written by a group of anonymous game designers, it declared that the "machinery of gaming has run amok", describing the elements of both the business and the practice of game development that conspired against creative freedom and risk-taking. Warren Spector, an industry-leading game producer at the time, received cheers and applause from fellow developers for declaring, at a conference in 2003, that "the publishers have to die, or we are all doomed" (Costikyan 2005). These feelings were widespread; sociologist Jennifer Whitson's 2012 study of game developers describes long-standing "alienation and disenchantment faced by many game developers", where creators are at the mercy of corporate interests (Whitson 2012, 156).

In terms of Cousins' analysis of the creativity problem, the warning he raises about the dangers that await those striving to innovate within the "design by convention" style domain of game design appear to chime with the lessons of Lawson's cartwheel example, referenced above. While a game designer may know that certain elements make for a good design, like the wheelwright,

they may lack understanding as to why. Game design innovation is either restricted to small, iterative adaptions (of the kind we tend to see in AAA games), or experimental, trial-and-error based production of small-scale games and prototypes (of the kind we see produced in the independent game development scene or at game jams).

3.6.4. The need for self-conscious design delayed by "brute-forcing" crafts-based design

The response to this states of affairs has not, as Cousins' hoped, been towards self-conscious design i.e. design drawing, supported by design tools. Cousins' 2004 vision of self-conscious game design practice has not been realised. No language of game design, nor any of the proposed versions of Doug Church's "formal, abstract design tools" (as discussed in Chapter 2) has been systematically integrated into mainstream game design practice. While particular studios may have their internal design languages and methods, they are not methods that are disseminated widely and they not do serve as a baseline or shared understanding within a larger design culture. Game design position descriptions attest to this: mastery of anything that could be considered "design drawing" is not a requirement listed among "required skills" for game design job candidates; rather, desired skills often include design-related production skills and, increasingly, "data-driven" design techniques i.e. the use of player metrics.

If anything, the reverse has occurred. Crafts-based design has become firmly entrenched. In the previous chapter we saw how, for example, data-driven design has formalised and – in some sectors of the industry – more or less enforced the crafts-based design method of iterative adjustments, adaptions and improvements.

3.6.4.1. Brute force compensating for inefficiency

But what of the inefficiencies of crafts-based design that result in a slow pace of design evolution, as highlighted by Alexander? In fact, recent changes in the game industry have meant that this is far less of a problem. These changes have allowed the industry to largely compensate for this inefficiency with what software engineers would call a "brute force" approach. The changes include significant changes in: 1) quality and cost of technology; 2) the number of games releases; and 3) the contribution of free and cheap design labour.

Firstly, since 2004 the technology supporting a trial-and-error style, crafts-based design process has significantly improved – the result being that in terms of labour and tools it costs less and

takes less time to both build games and test them. These changes are described in the previous chapter.

Secondly, there has been a significant increase in the volume and frequency of games being prototyped, produced and brought to market. We have only to compare the rate and volume of game releases before and after the launch of the Apple iPhone, for example. This is combined with the unprecedented scale of playtesting being carried out via online analytics. Data-driven design in games could be thought of as crafts-based design on a massive scale – as if thousands of wheelwrights were producing and testing millions of wheels.

Thirdly, just as pyramids can be built with inefficient engineering if you own enough slaves, game design progress and evolution can advance by the sheer number of designers working on design innovation, at no cost to the industry. In the previous chapter I described how changes to industry business models have given rise to changes in game design practice. Industry changes are again relevant here: they have meant that much of the risk burden of design innovation – and any costs and inefficiencies thereof - have been transferred away from publishers and investors, onto the shoulders of self-funded independent developers, as well as game design students and researchers (whose numbers have ballooned over the last decade). The price of design failure and experimentation is still very high, as it was in 2004, but it is my observation that now a significant part of the cost has been offloaded onto new entrants into the game industry: the games education sector, hobbyists, small developers, and the rise of a predominantly young, flexible, cheap or even self-funded labour force now offered to creative industries known as the "creative precariat" [14].

In this way, a remarkable increase in game design and production activity over the last ten years has, to a large extent, been able to serve as a substitute for deeper design understanding. Low design productivity is not a problem if the cost of design labour and technology is low. A decade of changes to the technology and industry context of game design have, at least temporarily, delayed or masked the need for a shift towards self-conscious design.

---

[14] Freelance, contract, self-employed, and intermittent workers in the arts, the media, and cultural industries (de Peuter 2014).

3.6.4.2. A cost to creativity

While inefficiency may not be such an issue, creativity arguably is. We have seen, in the previous chapter, how iterate-and-test style methods, while on the one hand offering designers more access to production to serve as the materials of design, could be considered more risk-averse than before. Risk averse because, like crafts-based design, it preferences piecemeal design moves, thus undermining, as we saw Whitson point out, the design mechanics of innovation and discovery – which is one of the major freedoms that self-conscious design aims to afford. The constant testing regime has the designer effectively surveilled, aware that any experimental design moves may have consequences external to design thinking.

For example, one of the software development industry's favourite aphorisms – now popular in all sectors of the game industry – is "fail early". In other words: build, test and, if necessary reject ideas or solve problems early. It is tempting to adopt this philosophy as a way of speeding up our slow, resource-intense crafts-based process. And yet, ambiguities and problems in the design situation (the importance of which I discuss in Chapter 10) are not necessarily failures, and are not always best resolved early in the design process.

The logic of data-driven design within the games-as-service sector in particular exerts pressure on a designer to think differently: rather than a designer making design moves within a consequence-free, safe space, in the context of private conversation with the materials, they are akin to a scientist whose actions must be motivated by empirical evidence and meticulously planned– all the while being conscious of the fact that even the smallest move may be subject to close analysis (A/B testing is used extensively in the games-as-service industry, for example).

3.7.   How to achieve self-conscious game design?

Within the context of game design practice described above, which elements could benefit from design support? Arguably a key weakness in game design practice is representation of the design situation.

Game design literature largely emphasises the notion that one of the important characteristics and difficulties designing games, as interactive media, is that it is a "second order" problem where designers can manipulate the results of their work only indirectly (K Salen & Zimmerman, 2003: 168). This means the results of their designs, short of making the game itself, are hard to

represent adequately. This is a problem, because representation of the design situation is required for both analysis and synthesis related activities.

Kuittinen and Holopainen confirm that game design practice is weak in terms of representation (Kuittinen and Holopainen 2009). Indeed, they find the lack of discussions of representation in the game design literature "perplexing", given how important representing is to game design. In view of the second order problem of game design, they argue, design activities associated with *formulating* (in Lawson's model) and with the forming of an *operative image* (in Löwgren and Stolterman's model) are crucial for game design. These activities are reliant on representation, and yet although skills associated with these activities seem to be emphasised by the literature, the discussion of representing is "very limited"; the consensus seems to be that designers should work mostly with two forms of representation: text and prototyping. The problems posed by prototyping being the only reliable form of representation have been discussed in the previous chapter.

We have seen that for self-conscious design to be enabled, we must enable the decoupling of design and making. For this to occur, designers must be able to represent the design situation without relying on making to do this. To echo Lawson's comment about the necessary emergence of self-conscious design in architecture, the separation between design and making "urgently requires" new ways of being able to model the final design.

As we have seen above, ways of modelling the design - new forms of representation - are offered by design drawing. It is design drawing that enables design thinking to take place around the representation of the design situation rather than in the designed product itself. Successful design drawing in games would allow the designer to formulate and manipulate at least some aspects of the design situation of the game without having to produce it. If tools could enable successful design drawing, therefore, they could be the key factor that facilitates self-conscious design in games.

Game design tools have the potential to take and have taken a variety of different forms: intelligent solutions that play the role of expert or colleague, pattern libraries that suggest solutions, card games that help brainstorm game ideas, and automated design that generates solutions. We can, of course, survey game designers for a wish-list of design support utility functions, and we can test the value proposition of tools in terms of their specific functions and

objectives against outcomes, and evaluate their applicability towards specific tasks. But there is a bigger picture to consider. While it is useful to look at individual elements and contributions, however, we should keep sight of a larger goal and consider how each piece of work fits into that goal's narrative. If our goal is to enable self-conscious design in games then we are framing our evaluation within a narrative of how each tool, like pieces of a puzzle, fits into the narrative of design drawing.

3.8.   The development of game design tools

It has not gone unnoticed by researchers and designers that game design process remains relatively underdeveloped compared to the sophistication of video game designs being produced today. As Stefan Grünvogel puts it:

> Game design as a craft has created a vast diversity of methodologies to balance interaction, game mechanics and audio-visual presentation for different game genres and players. But there are only very few attempts to support this process by using formal methods (Grünvogel 2005).

Game developers themselves have made similar observations:

> Compared with the vast body of operational knowledge found in the world of filmmaking, the game design community is just beginning to articulate the concepts and techniques specific to our medium in order to establish methods of game design (Kreimeier 2003).

Academic and industry discourse has developed to highlight this absence in the field of game design, and specifically the lack of concrete and conceptual tools for game designers. Researchers and designers have noted that we lack game design support in the form of computer-aided design (as opposed to production) software, and formal or semi-formal design models and concepts (as distinct from heuristic approaches) that can support game design tasks. Nelson and Mateas observe that game designers "have no tools for reasoning about and visualizing systems of game mechanics" (Nelson and Mateas 2009). Agustin et al claim that this lack of methods and tools to help game designers support the 'ideational' stage of game creation has contributed to a lack of innovation in commercial game design. This lack of creative risk-taking can be linked to the cost of creative mistakes (in terms of time and resources) in a craft that relies on production to support design. Even "rapid" game prototyping can require not insignificant time and resources as well as production expertise.

3.9.   Explaining the lack of progress towards design tools in game design

The game industry has no shortage of software-based tools. The video game industry has been described as "a production oriented industry" where the majority of the technology has focused on production tools." (Agustin et al. 2007). Game artists, game programmers, level designers, quality assurance testers and project managers all use specific software – sometimes developed 'in-house' for the purposes of a single game project – tailored for doing their job. In an industry that has a strong tradition of developing bespoke tools, one has to wonder why software tools to support game design tasks are not built and widely used.

Seeking an explanation for this leads one to deeper problems within the field of game design. Game designers are not yet applying even purely conceptual "on paper" design tools or graphical notation systems to their design work. These conceptual tools and systems, evolved and confirmed in practice, would form the basis for any computer-aided design support software. (If architects, for example, had not yet devised a way to draft on paper then there would be little point in developing CAD technologies.) Given this poor state of disciplinary evolution, it is no surprise that the primary tool of a game designer is commonly a word processor. Outside of playable contexts (a prototype, for example), the only means we have of modelling and communicating gameplay concepts has been natural language; we do not yet have a shared and commonly understood framework for designing games with anything other than words.

For certain elements of a game – narrative, character design, high-level concepts, for example – descriptive prose and illustrations (storyboarding, for instance) may be serviceable. For a key component of a game design, however, it is not. Defining this key component requires breaking down games themselves into their component parts. Salen and Zimmerman offer three sets of schema that can be used to frame games: rules, play and culture. Rules are the formal elements; "the inner, essential structures that constitute the real world objects known as games (K Salen and Zimmerman 2003: 80). Of all possible schema, the formal, systemic, structural aspects of a game design are arguably the least well served by natural language. This is problematic given that these are the aspects leading theorists consider the defining characteristics that set video games apart from other media. "A game is a system … defined by rules..." (K Salen and Zimmerman 2003: 80).

As Game Studies has developed an orientation to games as systems, it has begun to notice that the lack of a formal means of communicating the mechanics of these systems hinders both game analysis and design:

> Ludology, the study of games in general and videogames in particular, has pointed out the need to create models in order to explain the mechanics of games. This lack of a notation to precisely define games and game mechanics has been a traditional game design problem (Reyno and Cubel 2009).

Even the language designers use has been criticised for not being as formal, standardised and precise as it could be. In 1999 designer Doug Church complained that we lack even a common vocabulary of terms to describe game concepts (Church 1999). He added that this is a serious problem if we want to pass down and build upon knowledge from generation to generation of game designers, a lack of a common design vocabulary being "the primary inhibitor of design evolution". Educator Tracy Fullerton, in her game design textbook *Game Design Workshop*, also complained of this some years later, calling the lack of a single vocabulary "one of the largest problems facing the game industry today" (Fullerton 2008: 40).

But even the design concepts and techniques that would form the building blocks of any such language are still in the process of being formalised, standardised and shared. Game design has for a long time been a kind of "dark art". Designer Dan Cook goes as far as to label past game design achievements as "accidental successes". He writes: "We currently build games through habit, guesswork and slavish devotion to pre-existing form." (Cook, 2007) A 2003 article by Kreimeier surveying the state of the art of game design method criticised game design texts of the time. These, it was argued, were so informal as to warrant being called "a kind of 'Discourse by Anecdote'", in which "game design experience is presented as a narrative, e.g., as a series of anecdotes and invented dialogs, sometimes as recommendations derived from interviews, or simply as annotated transcript" (Kreimeier, 2003).

Raph Koster is another designer who has argued for formalisation. In an influential presentation at the Game Developers' Conference (the industry's principal conference for developers) in 2005 entitled "A Grammar of Gameplay" (Koster 2005), Koster highlighted the imprecision of natural language as a tool for designing gameplay and urged designers to develop a graphical notation system for game design. Two years later he repeated this call in an interview, saying "we want it, because god damn do design documents suck as a means of communicating game design" (Sheffield, 2007).

Designer Ben Cousins complained in 2004 that compared to other design disciplines, game designers lack formal training in their craft:

> …the only difference here between other media and games is that every moviemaker, songwriter, painter and novelist is acutely aware, and often trained in, the application of the appropriate primary units. Game designers have not yet moved into that phase (Cousins 2004b).

Given this lack of expressing design ideas with any level of formality or abstraction, the goal of developing a software tool for game design seems akin to that of developing music notation tools for composers who cannot read music.

3.10. Formalisation towards models and tools

Over the last ten to fifteen years the "discourse by anecdote" has begun to be replaced by some attempts towards comprehensively defining, analysing and describing these technical game design concepts that we lack. Researchers and designers have also attempted more experimental work in the form of proposals for formalised, and sometimes visual methods of modelling and describing gameplay. Some have been explicitly conceived to be used as conceptual tools for game design.

> From Doug Church's 'Formal Abstract Design Tools' (1999) to Björk and Holopainen's 'Game Design Patterns' (2004) we have a wealth of frameworks that seek to develop a unified discourse among designers, to promote clarity, better game design, and a clearer procedural structure for designers in the creation of their games(Bojin 2010).

These frameworks range from "semi-formal approaches" (Grünvogel, 2005) or "textual interpretations of game design practices" (Koster, 2005) to proposals for graphical modelling systems. Approaches developed by researchers include *Patterns in Game Design* (Björk&Holopainen, 2005), Järvinen's "library of game mechanics" (Järvinen 2008), Hunicke, Leblanc, and Zubek's "Mechanics, Dynamics, Aesthetics" framework (Hunicke, Leblanc, and Zubek 2004). Even Sicart suggests that his definition of game mechanics – summarised as "methods invoked by agents, designed for interaction with the game state" - be used practically "as a formal tool for describing and modifying mechanics in a coherent and comprehensive way" (Sicart 2008).

3.11. Conclusion

Design studies offers insight into current game design practice, and allows us to compare game design against design practice more widely. We can see that in the domain of games, where design

by making is considered best practice, design is, arguably, an "unselfconscious" or crafts-based practice.

In games we lack the means to separate design from making. For this, a designer needs to be able to represent the design situation so that design thinking can occur in the representation instead of where it is mainly occurring now – in a playable version of the game. To represent the design situation separate from making, a designer uses "design drawing".

In game design we lack forms of representation, as well as design drawing tools with which to create them. This lack is probably in large part attributable to the nature of games themselves: games are difficult to represent - game design being a "second order problem" where the designer only indirectly affects the outcome of their design.

To satisfy this need to separate design from making, attempts to create game design tools have been made, many of which I have reviewed here. Design tools for games propose to offer game designers a means of design drawing: representing and allowing manipulation of the design situation. If we see game design tools in this light – i.e. as design drawing for games that could enable game design practice to move from crafts-based into self-conscious design - we can see the significance of the work on game design tools, as well as the qualitative nature of the change that the adoption of such tools into practice would engender.

4. GAME DESIGN TOOLS

4.1. Summary

In this chapter I give a more developed definition of game design tools, before reviewing the tools the research community and industry has produced that fall within this scope. I then compare and discuss the tools in relation to the types of design activities they support.

4.2. A definition of game design tools

For the purposes of my research I define a "game design tool" as:

A **software-based or conceptual tool**, the **primary function** of which is to support game designers in **design thinking**.

Below I break down each element of this definition and describe it in detail.

4.3. A design tool

A design tool is a structured form of support that is intended to aid the design process. Most importantly, it allows a designer to engage in some of the vital activities of the design process that aid design thinking.

In this definition of design tool I am not including tools that are dependent on the production of a version of the game. While the gathering and analysis of metrics (i.e. the materials of data-driven design) and automated testing support design, they are dependent upon production as part of an iterate-and-test design approach.

The definition of "tools" in the discussion around game design tools varies widely: from "tool" in a rather broad sense to software tools. In Doug Church's 1999 article "Formal Abstract Design Tools", for example, he proposed the need for "tools" as a "shared language" of design concepts derived from previous game designs that designers could use to apply to their own design problems (Church 1999). Since then, the solutions proposed by practitioners and researchers have been quiet diverse: including, for example, taxonomies of design concepts, diagramming methods and artificial intelligence based semi-automated design support software.

4.3.1. Design tool metaphors

Design support researchers have identified the fact that the addition of computation to a tool can challenge our notion of what a tool is. Game design tool designers have also looked at this question. They enlist metaphors to frame discussions of the roles computational design support might perform, and what the place of technology is in the design relationship or conversation.

Lubart, in relation to design support in creative domains, uses the metaphor of the tool performing the role of "nanny". The nanny helps plan and takes care of tasks in order to "free up users' minds". The "coach" proposes information about existing techniques, provides "diagnosis-assistance modules" and guides the designer towards new ideas. The "colleague" role contributes new ideas as part of a dialogue with the designer, suggesting novel, unconventional ideas that the designer has not thought of. Computers are especially suited to this, suggests Lubart, because they are better at implementing random searches than humans (Lubart 2005).

Gillian Smith et al., in discussing game design tools, introduces "collaborator" – a refinement of this "colleague" concept. She suggests we might aim to create game design tools that raise the role to become more of an equal partner, able to bring the designer's conversation with the materials to life as an active participant, debating and even requesting changes to decisions made by the designer – as a human collaborator would (G. Smith, Whitehead, and Mateas 2011a).

Towards a better understanding of the role or roles procedural content generation (PCG) can play in games and game design, Khaled, Nelson and Barr have proposed metaphors: "tool", "material", "designer", and "expert", which they describe as a set of lenses through which to view PCG in its varying contexts. "Designers" are PCG algorithms, encoded with expertise in aspects of game design, are tasked with solving game design problems and conducting design tasks with little or no intervention from a designer. "Experts" supply external expertise to be taken into account by a human game designer. "Tools" are devices or instruments manipulated for the purpose of achieving specific design goals, enhancing and extending the designer's abilities. "Materials" are procedurally generated substances that can be reshaped by the designer and deployed in the game. To generate them the designer articulates a set of property constraints that describe a class of potential materials that are desirable (Khaled, Nelson, and Barr 2013).

Within Khaled et al's framework, I am covering technology that serves as tools – not so much experts or designers – including tools that allow me to create materials.

To make sense of these metaphors, we could see them in the context of an evolving understanding of Computer-Aided Design (CAD) over several decades, which has been used as a starting point for developers of game design tools. Nelson et al traced the history of CAD tools in architecture and design for the purpose of understanding how we should build such tools for game design.

Early work in CAD tools, which date back to 1956, saw CAD as having a dual role: offering a front end for graphical modelling, and a backend that performed automated reasoning for the role of advisor – i.e. enhancing the back talk of a situation, illuminating constraints and implications. The second wave of CAD tools focused on the importance of knowledge in specific design domains. Developments in the 1980s brought knowledge-based AI systems, which enabled tools to offer back talk based on domain-specific knowledge. Then domain-oriented design environments (DODEs) extended the idea of domain-specific knowledge to include not only factual knowledge but also design knowledge - i.e. best practices and common solutions. They critique designs, provide design suggestions and reasons for them. This shifted the computer's role in the design conversation from providing back talk to actively participating in the design activity (Nelson and Mateas 2009).

Overlooking any subtleties for the sake of simplicity we could see at least 3 broad types of roles, rising gradually in importance and status. First, the "designer's slave" "nanny" "assistant" type functions, where the tool is commanded to use its computational power to help increase efficiency and relieve cognitive load, help with planning and organisation tasks. Then, the "advisor" role, where the tool helps give the designer insight into their design, providing information it has deduced based on the content of the design situation, thereby enhance the back talk of the design situation. The "coach" "colleague" and "expert" even have ideas of their own, bringing in knowledge from outside of the design situation to shed light on what the designer has made, and even offering new ideas and suggestions for improvements. Completes this ascent from slave to peer, the "collaborator" has the power to make design moves of its very own. Finally, the computer as "designer" becomes the dominant collaborator in the working relationship, with the capacity to design games all by itself.

4.4. Including conceptual tools

As well as software tools, I am including conceptual tools in this study.

In Chapter 4 I argued that the lack of software or more sophisticated tools for game design can in large part be attributed to a lack of visual notation upon which to base such tools. Unlike more developed domains such as music, which can base their digital tools on existing formal systems and visual notation, games do not yet have an accepted standard for a shared visual language. The success of the formalisation of game concepts into visual terms is therefore key to the success of the tool that uses it. Furthermore, according to the scope of the definition of 'tool' for this study, a visual language is a form of tool; some of the tools looked at in this study are a more or less visual languages or tools for authoring language. Dormans' *Machinations* was originally created as a diagramming language, for example (J Dormans, 2012: 201)

A conceptual tool for my purposes is fairly broad. It is any form of structured, semi-formal method or model – a model being a language or framework that allows the designer to define relationships between concepts – that can be used as part of a game design toolset across multiple (but not all) games and game genres.

Distinguishing which aspects of the conceptual work on game analysis and design could be considered "design tools" - in the sense they are concepts practically and reasonably directly applicable to design – is not straightforward. To make such a distinction for the purposes of my research I have enlisted a schema from Design Science Research: constructs, models, methods and instantiations. Constructs form the vocabulary of a domain; a model is a set of propositions or statements expressing relationships among constructs; a method is a set of steps or guidelines used to perform a task; and an instantiation is an artefact that has been realised using constructs, models and methods (March & Smith, 1995). The frameworks in this area of game design research contain elements of constructs, models and methods to varying degrees. I have chosen to focus on models – i.e. tools that give game designers a way to create and describe relationships between elements (i.e. "constructs") of their design, rather than heuristic, 'how to' or 'rules of thumb' style methods for game design.

Much of the work on constructs and methods is still useful to have in view, however. As game design lacks standardisation in core concepts and vocabulary, the constructs used tend to differ between models, and for that reason it has been useful to include work on constructs in my purview also. Methods are less relevant, but they do represent the current state of the art, dominating game design literature; thus they provide a certain degree of context for this work and are referenced where appropriate.

According to Salen and Zimmerman a "game design model" is a kind of systemic thinking generalised to cover many or all games. It provides a vocabulary and a set of concepts for thinking about games and for solving problems as they emerge in the design process (Salen and Zimmerman 2003). Somewhat contradictorily, and possibly due to the fact that they found it difficult to find many examples from real-life practice that fit this definition to serve as case studies, they broaden the definition to include models that designers have devised specifically for one game they worked on.

> Each case study analyses a game by describing the rule-structures unique to the game, creating a formal model specific to the individual game at hand (Katie Salen and Zimmerman 2005:55).

By this definition then certainly, most expert game designers could probably be said to use semi-formal approaches to a greater or lesser extent. For me, however, this rather undermines one of Church's key goals: tools that can be shared across many games, designers and, to some extent, game genres. Likewise, my definition is narrowed to looking specifically at conceptual tools that attempt this challenge.

4.5. Design support as primary function

I am including only those tools that have supporting design thinking as their primary goal and function. In Chapter 9, I break down this goal into more specific functions, attributes and criteria for game design tools that I use to analyse and evaluated my experiences with the tools under study.

Tools where the focus is primarily on production are outside this scope. This includes the designer-friendly game development tools such as *Game Maker* or level editing tools that are sometimes used by designers as for design thinking purposes. As I have suggested in previous chapters, the absence of tools created specifically for game design is arguably one reason designers lean heavily on production tools for design, using them – even if imperfectly - in exploratory, sketch-like ways. There is also another way in which production tools can serve as design tools: in the sense that production itself can be a design activity, powering design-by-making within crafts-based practice – which, as argued in the previous chapter, dominates contemporary game design. This context is important, as my approach with this project is very much about taking current practice as the starting point, and taking steps toward conscious design from that point. I am not looking at design tools from the point of view of their suitability for immediately replacing design-by-making

so much as whether they might meaningfully add to the game design toolset. So while production tools are not specifically addressed in my research, I have continued to use conventional design-by-making methods where appropriate, alongside my work with the design tools. This is explained further in the following chapters.

4.5.1. Simulation as a "sketch" rather than a prototype

Though superficially similar in some cases, we can distinguish modelling game designs in design tools from prototyping.

Some of the game design tools under study offer a visual representation of the player experience – in effect, allowing the designer to see the dynamics of the mechanics they are drawing. In the design research literature, creating a graphical representation of a design situation is sometimes described as "sketching". Accordingly, Nelson, Smith and Mateas have used the term "game sketching" in reference to their design tool work, for example.

Though Manker and Avola observe that the game prototype is often used as a sketching tool in game design (Manker 2011a) the term "game sketching" allows us to distinguish the activity from prototyping. Manker notes the useful distinction made between prototyping and sketching in the field of User Experience or Interaction Design (Manker 2011b). Where prototypes are used to evaluate a design, often called an "experience prototype", "the audience of a prototype is supposed to be actively participating when using it" (Buchenau et al, 2000, cited by Manker, 2011). Applying this to games, we could say that when a designer interacts with the prototype they assume the role of, and imagine themselves as, a player. A response from a designer that Manker interviewed seems to align with this idea, in saying that a prototype allows one to "feel" the game. In the case of a sketch, by contrast, the designer is not so much aesthetically and emotionally engaged so much as engaged in design thinking; the simulation in a sketch serves as a representation of the design situation, allowing the designer to "see" the logic of their design, and offering the back talk necessary to a design conversation. Simulation and interactivity here could be thought of as extension of visualisation, a product of design drawing that enables the designer to "see" and evaluate the logic of their design – as a designer sees it (this is idea is discussed further in Chapter 10).

In addition, the speed of sketching as compared with prototyping is important. A sketch is made rapidly, so as to quickly express an idea, or serve as a contribution to a design conversation – with

other members of a team (Agustin et al. 2007), but also, presumably, to enable a Schön-style conversation between the designer and the sketch itself. If we recall Schön's seeing-drawing-seeing loop (see Chapter 3), we can understand how important the speed element of a sketch is.

Nummenmaa, Kuittinen, and Holopainen discuss the benefits of a simulation created in a game design tool, which they define as similar to yet distinct from prototyping. Unlike game prototypes, simulations can reveal problems and opportunities over whole game, rather than just the immediate and short term dynamics that a prototype models. In other words, simulation allows a designer to view system behaviour at a higher, "big picture" level than a prototype would comfortably allow (Nummenmaa, Kuittinen, and Holopainen 2009).

4.6.   Supporting a *game designer*

Word processors, spreadsheet software (e.g. *Microsoft Excel*) and flowcharting tools (e.g. M*icrosoft Visio*), paper, whiteboards – these are tools that can be and are used by game designers to help them communicate and conceive their designs. But they are not tools specifically developed for the purpose of game design and hence are not being evaluated in this study.

4.6.1.   Tools that blur the role between designer and tool

I am covering designer-centric procedural content generation tools, or "mixed initiative design" (described below). What I am not including are PCG technologies that require implementation by programmers within a playable game – i.e. design-by-making – for design thinking to occur. I am also excluding automated design tools where the focus is to enable design by non-designer – i.e. what Khaled et al describe with their metaphor "designer". Tools such as *Game-o-matic* (Treanor et al. 2012), for example, do not fall under my purview.

I am also compelled to exclude very specific or future design tasks (e.g. the possibility of design tools for character AI (Hecker et al. 2009)). These directions will be interesting to explore in future work, but at present there are no tools accessible to me of this type.

With the fast moving evolution of game design in mind, however, it is useful to acknowledge that the design definitions I am using are imperfect, in that they adhere fairly closely to conventional understandings of what a design tool is. This is because those definitions may be changing, and ultimately be changed by game design itself. The impact that these new technologies, in terms of

both tools and materials, could have may force us to expand or revise our understandings about what design tools are, or even what it means to design.

4.7. Tools within this scope

Here I review formal game design tools and models developed over the last fifteen years. It comprises both conceptual and concrete tools.

Categorising tools by their design support approaches is not straightforward, as often more than one approach can be seen in the same tool. For example, the *BIPED/Ludocore* system uses both resource flow and system modelling, and the *BIPED* front end offers diagramming.

4.7.1. Models

4.7.1.1. Taxonomies

One category of work towards formalisation to aid design has involved defining the elements of game design into taxonomies and schemas. When Doug Church proposed the idea of "formal abstract design tools" he gave examples that loosely resembled Alexandrian patterns used in software development. By "tools" Church means conceptual tools: design concepts that can be applied over and over again to similar design problems in different games. Since Church's article, there have been several attempts to come up with ways to create schemas and taxonomies of this kind that are formal, but still described in natural language. They have been described as "semi-formal approaches" (Grünvogel 2005) and "textual interpretations of game design practices" (Koster 2005).

One is Bjork and Holopainen's framework for and collection of *Patterns in Game Design* (Björk and Holopainen 2005). When surveying the domain, Salen and Zimmerman found this work to be "the best example we have found of Church's formal abstract design tools" (Katie Salen and Zimmerman 2005:55). The game design patterns are inspired by Alexandrian patterns, and follow a similar format. Each pattern defines a game mechanic, and most importantly, which game design problems it can solve and how it can work in combination with other game patterns or mechanics.

Another taxonomy, devised by Järvinen (Järvinen 2008) comprises formal definitions of game elements and a "library of game mechanics". Järvinen describes his library, however, as "analysis-oriented – with design consequences if one so desires". Initiated at the Game Developers'

Conference in 2001 by designer Noah Falstein, *The 400 Project* is a set of methods for game design, styled as "rules of thumb". Falstein and Barwood describe it as "an attempt to write up 400 rules of game design that can be used by designers to make better games" (Falstein and Barwood 2006).

4.7.1.2. Unit-based models

The game design community has developed a discourse around unit-based models and notation systems. Several designers have proposed devising a 'grammar', the building blocks of which are to be found by analytically breaking down games to their core units at the lowest level of structural granularity; in other words, "any single conscious interaction that cannot be further subdivided" (Cousins 2004b). These core elements are moment-to-moment player decisions or actions, which have been described by various theorists using metaphors that reference linguistics and chemistry: "verbs"(Crawford 2002); "ludemes"; "atoms" (Cousins); and "choice molecules", with the last of these defined by Salen and Zimmerman as "action > outcome unit" which is "at the heart of interactive meaning", and from out of which larger interactive structures are built" (K Salen & Zimmerman, 2003: 63).

Cousins, the game designer who coined the term "game atoms", compares game atoms to similar concepts of "primary elements" in other art forms, for example, notes in a piece of music, shots of a film, and brushstrokes of a painting. He claimed that "every artistic, emotional, great feeling moment you have ever had playing video games was made up of primary elements, whether the designer acknowledged and specifically designed them or not" (Cousins 2004b).

These notions of granularity support a reductionist, hierarchic, layered approach to analysing or constructing a design. Cousins describes the role of the concept as focusing the designer's attention on the lowest layers of their game so that they can decide which kind of feedback is appropriate to give at a given structural layer (Cousins 2004b). Similarly, Bojin argues that as units of analysis, game atoms "promote the critical examination of the design of a game from the player experience down--drilling into the specifics of what contributes to those experiences" (Bojin 2010). Designer Tynan Sylvester believes that game atoms can help a designer imagine the playable outcome of their design, arguing that it is essential that one must fully understand the smaller elements that make up the system and how they interact before one can fully predict the overlying system (Sylvester 2005).

Designer Raph Koster extends this to relationships between the atoms or elements. To him the atomic model "allows us to examine logical links and feedback loops clearly" (Koster 2005).

4.7.1.3. Resource flow

The resource flow model is based on the theory that game systems can be expressed as economies, and core game mechanics can be reduced to the flow of resources. A resource is anything that can be "created, moved, stored, earned, exchanged, or destroyed" (Rollings and Adams 2003). Interestingly, this is the only conceptual model that has been reified in the form of a software-based tool (*Machinations* (Dormans 2009)).

4.7.2. Brainstorming aids

A number of researchers have produced sets of cards to be used for teaching and brainstorming game design or specialist areas of game design. In some cases they are intended to serve as a learning tool or aid for non-expert practitioners (e.g. (Alves 2013)), or for professional as well as academic purposes (e.g. (Järvinen 2005)). Typically, the decks typically contain game elements (e.g. (Järvinen 2005; "GAME SEEDS" 2015)) or patterns (e.g. (Wetzel 2014; Alves 2013)) and are accompanied by a set of rules that structure and constrain the learning or brainstorming within the context of group play.

4.7.3. Diagramming

Both practitioners and researchers have proposed the development of a diagramming language as a tool for game design and analysis. Towards this goal, some attempts have been made to develop notation and graphical modelling systems. As mentioned previously, diagrams are already commonly used in game design documentation to describe certain elements of their design. There is no standard model for this, however. Further, Araújo and Roque observe that the diagramming currently used by designers, commonly to describe high-level game logic, typically fails to describe a game's underlying system, and "prevents designers from using these diagrams in a more effective way, such as for finding balancing issues or problems within the game's flow" (Araújo and Roque 2009).

4.7.3.1. Koster's game atom-based notation system

Koster proposes that game atoms could form the basis for the development of a notation system for game design. Such a notation method, he argues, could help designers more easily detect

design flaws like degenerate strategies - in the way that a composer can look at their manuscript and quickly spot parallel fifths; and emergent behaviours. Advocating the use of graphical notation, he argues that "iconic descriptions would force us to examine the actual space" (abstract space, we should assume) of our designs (Koster 2005). Towards this end, Koster uses his atomic model as the basis for a diagramming language. An example of this can be seen in Figure d, a diagram of the game Checkers (Draughts) made by Koster using his notation system.
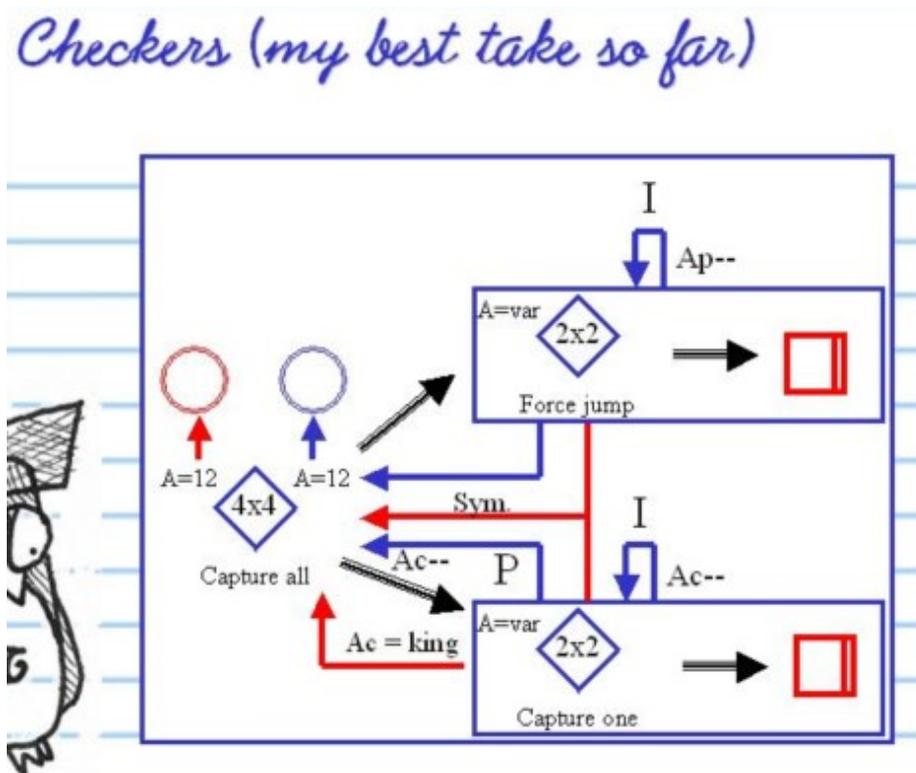
Figure d: An example diagram built by Koster using his own notation

4.7.3.2. Diagramming with non-domain-specific models

This work has drawn on existing paradigms such as UML (Sicart 2008) and Petri nets (Araújo and Roque 2009) (Natkin and Vega 2004) (Bura 2006).

4.7.3.2.1. UML-inspired models (UML)

UML (Unified Modelling Language) is a modelling language used by software engineers to model their software designs. Sicart proposes that designers could model game mechanics using the modelling language UML, using his formal, object-oriented-programing-inspired definition of game mechanics as "methods invoked by agents, designed for interaction with the game state". Other

researchers have proposed UML-inspired modelling for game design purposes too, such as Taylor, Gresty and Baskett with "game-flow diagrams" (Taylor, Gresty, and Baskett 2006).

The examples often given to demonstrate these diagramming approaches are very much about modelling game flow or player decision paths – similar to business process modelling - rather than the core mechanics of the game system itself as a complex system. This raises the question of how useful this kind of diagramming could be to a designer who wants to model the core game mechanics. In the case of "game-flow diagrams", the researchers have explicitly acknowledged that their model is limited to the design of scripted story-driven gameplay rather than games with systems that generate less predictable gameplay dynamics. Unfortunately, in practical terms designers need little help outside their traditional toolset to design such story-driven games.

4.7.3.2.2. Petri nets

A Petri net is a mathematical modelling language for the description of distributed systems. Diagramming using Petri nets has been proposed as a game design concepting method (Araújo and Roque 2009) (Natkin and Vega 2004). This has been described as a combined formal and graphical approach (Grünvogel 2005).

Figure e shows a an example diagram made by Araújo & Roque the design of a pit stop tyre-changing sub-system in a car racing game. As they describe it:

> …we have a situation where a F1 car is making a pit stop to change all of its tires. When the Stop transition fires, the places representing the tires – front and rear, left and right – receive a token. From that point on, all the tires are changed concurrently. The Go transition can only fire when all the "Tire changed" places that are connected to it hold a token (logical AND). This means that the car can only leave the pit when all the tires are in place.

Stéphane Bura proposed a grammar inspired by Koster's grammar, as well as Petri Nets and his experience with cybernetics. Figure f shows a diagram for the card game Blackjack. He proposes that abstract elements such as "skill", "luck" and "risk" could be re-used from one game to another (Bura 2006).

63
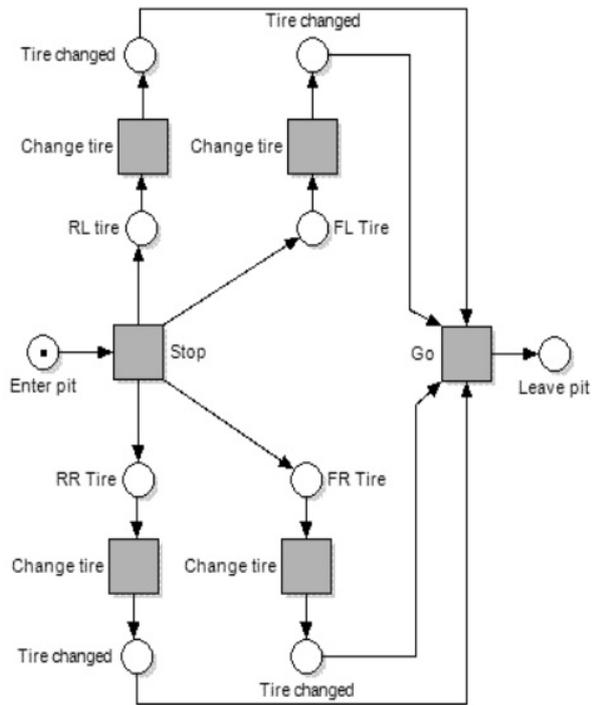
Figure e: A sub-system of a game modelled in Petri nets by Araújo and Roque (2009)
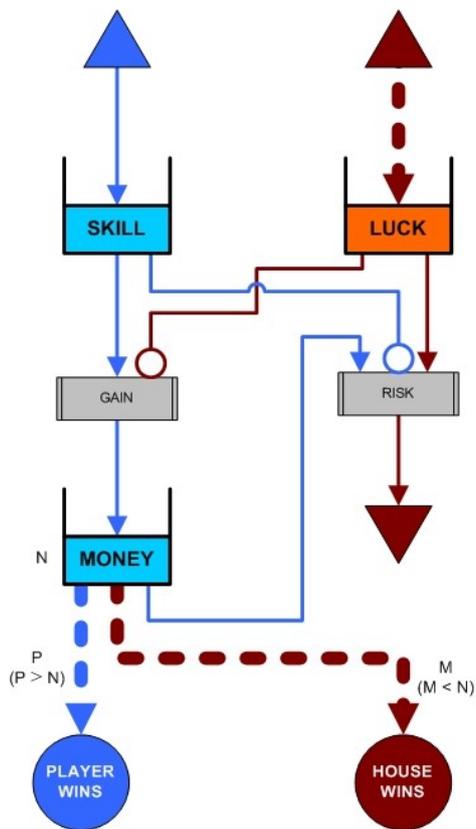
Figure f: Bura's diagram of the card game Blackjack using his own diagramming method (Bura 2006)

4.7.4. Game sketching

4.7.4.1. Sketch-It-Up!

Created as part of a post-graduate research project at Carnegie Mellon University, *Sketch-It-Up!* (Karakaya et al. 2009) is a set of processes and technologies designed to be used at the ideation stage of game design. This work builds upon an earlier "game sketching" system developed by Agustin et al (Agustin, Chuang, Delgado, Ortega, Seaver, and Buchanan 2007a). *Sketch-It-Up!* appears targeted towards the design of linear, narrative-based games: it supports the modelling and rehearsal of game flow at a high level: specifically, the number, order and difficulty of game interactions and the rewards awarded to the player.

4.7.5. System modelling and automated reasoning

4.7.5.1. BIPED and *Ludocore*

BIPED is a system that supports the early stages of game design (A. M. Smith, Nelson, and Mateas 2008). It is powered by *Ludocore*, a "logical game engine" designed to support the modelling of game systems and simulate their dynamics. Both *BIPED* and *Ludocore* evolved from earlier work in generative and automated design for games, undertaken within the context of artificial intelligence research. This work has previously been described as a "game design assistant" (Nelson and Mateas 2008) (M.J. Nelson & M. Mateas, 2008b) and a "game sketching" tool or language (A. M. Smith, Nelson, and Mateas 2009).

For its game sketching language, *BIPED* takes its cue from the abstractions used in paper prototyping: e.g. cards, tokens, and dice. Such interface elements can be seen in the game model made for *BIPED* shown in Figure g. *Ludocore* performs player modelling and provides designers with the ability to query potential consequences of rule interactions. In other words, it can provide a designer with answers that could otherwise only be gleaned by building a version of the game and using extensive human or automated testing.
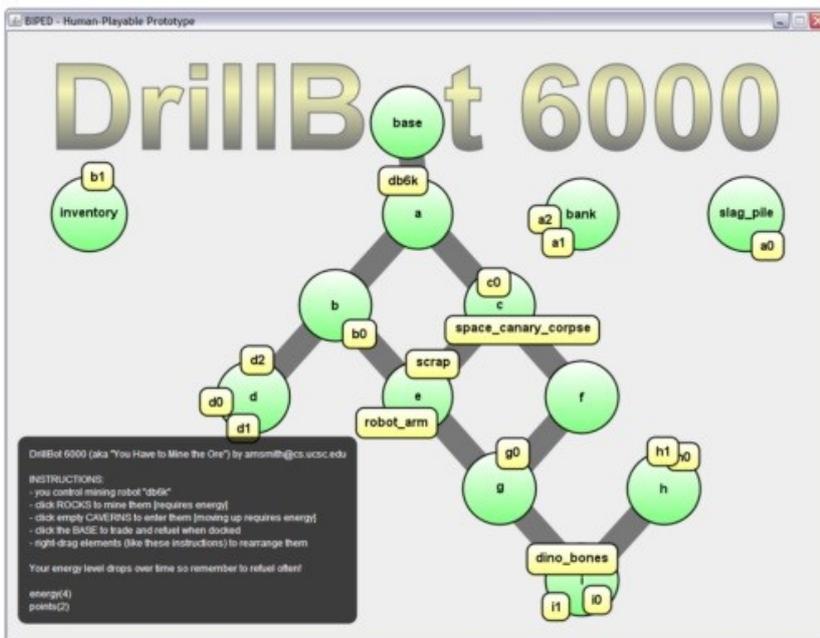
Figure g: *BIPED* tool (A.M. Smith, M.J. Nelson, & M. Mateas, 2010)

4.7.5.2. Machinations

Joris Dormans' tool *Machinations* affords game designers a means of communicating and modelling "the structure of game systems and patterns that might be found in these structures" (Dormans 2009). It extends Adams and Rollings' "resource flow" model (Rollings and Adams 2003) (see Resource Flow model later in this chapter), implementing it in the form of a Petri-nets-inspired real-time modelling and simulation environment. Its graphical editor allows a designer to model their game system and 'run' the simulation in real-time or faster than real time, revealing emergent dynamics of the system over time. As a tool, it answers LeBlanc's emergent dynamics and feedback. These dynamics can even be visualised via graphing components. Figure h shows a Machinations diagram that models the travel and trading system in the game Elite. Dormans describes it:

> The player's location on planet A or planet B activates the converters that implement the trading mechanisms in the center. A few possible ship upgrades are included on the right (Dormans 2012).
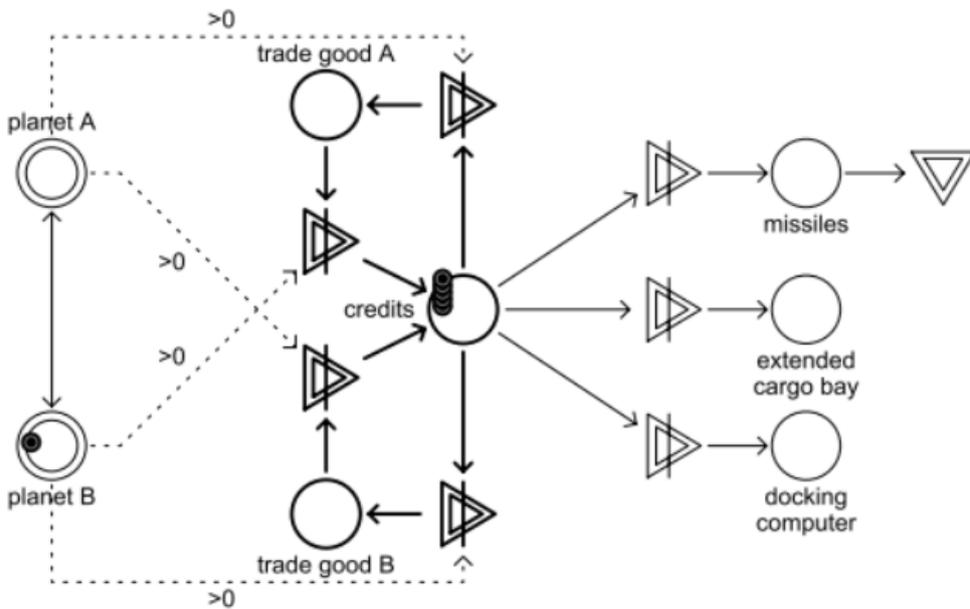
Figure h: A *Machinations* diagram showing the trade and travel system from the game *Elite*.

4.7.6. Mixed-initiative level design

Mixed-initiative design is an approach currently favoured by researchers in the domain of procedural content generation for games. The goal is to improve upon current procedural content generation techniques that allow little designer control or input. "Mixed" designates the involvement of both the computer and the designer, the pair entering into a design conversation, each contributing to the design, with the designer guiding the creation and constraining the design space (G. Smith, Whitehead, and Mateas 2011a).

4.7.6.1. Tanagra

Tanagra is a mixed-initiative tool for the design of levels for 2D platformers, aimed at reducing authorial burden in level design while still allowing human designers to exercise their creativity and aesthetic judgment.

A "reactive level generator" creates multiple levels according to specifications provided by the designer. The designer can choose from these and also make refinements to the level, including placing and moving level geometry. AI techniques of reactive planning and numerical constraint solving ensure that levels are playable (G. Smith, Whitehead, and Mateas 2011b). Figure i shows *Tanagra*'s editing interface.
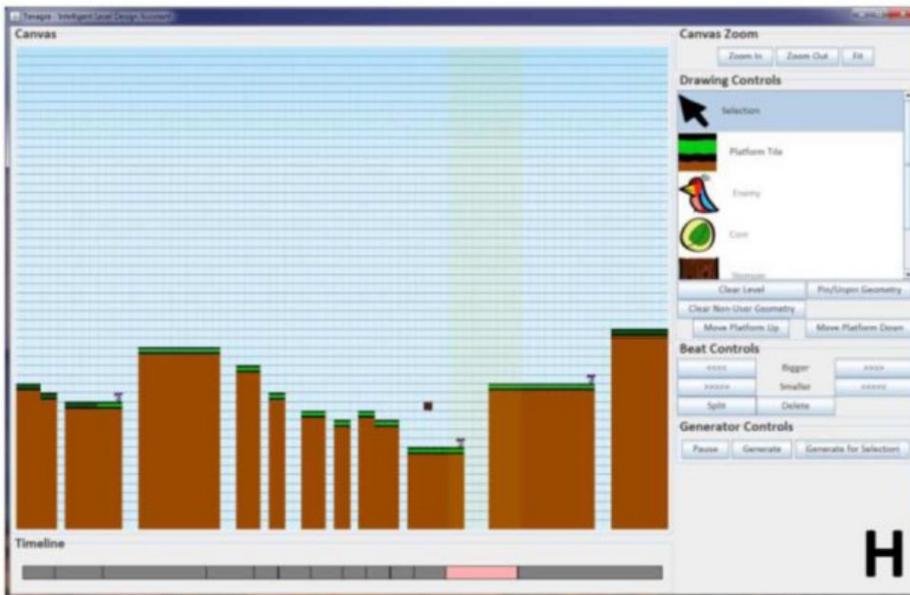
Figure i: *Tanagra* a "reactive level editor".

4.7.6.2. The Sentient Sketchbook

The Sentient Sketchbook (Liapis, Yannakakis, and Togelius 2013) provides support for level design in the design of strategy games. Specifically, it enables the designer to ensure the playability of the spatial design of the map (level), and aims to reduce the effort involved in producing a detailed map.

The designer is afforded an interface that allows the designer to create abstract, low-resolution "sketch" of a map containing player bases (starting positions), obstacles and resource locations. It automatically evaluates the map and displays information on how the map aligns with playability constraints – in particular, how equitable access is to resources between opponents – and calculates and displays navigable paths. In addition, Like *Tanagra*, *The Sentient Sketchbook* aims to approximate the role of a design "colleague" it uses generative algorithms to add details and suggest novel alternative designs. These are updated in real time and use the designer's current working map as a starting point. The algorithms attempt add details that create an organic looking map, while retaining the fundamental gameplay properties of the map. Figure j shows the interface for *The Sentient Sketchbook*. The designer edits their strategy game map on the left. The maps on the right are suggested map layouts made by the tool.
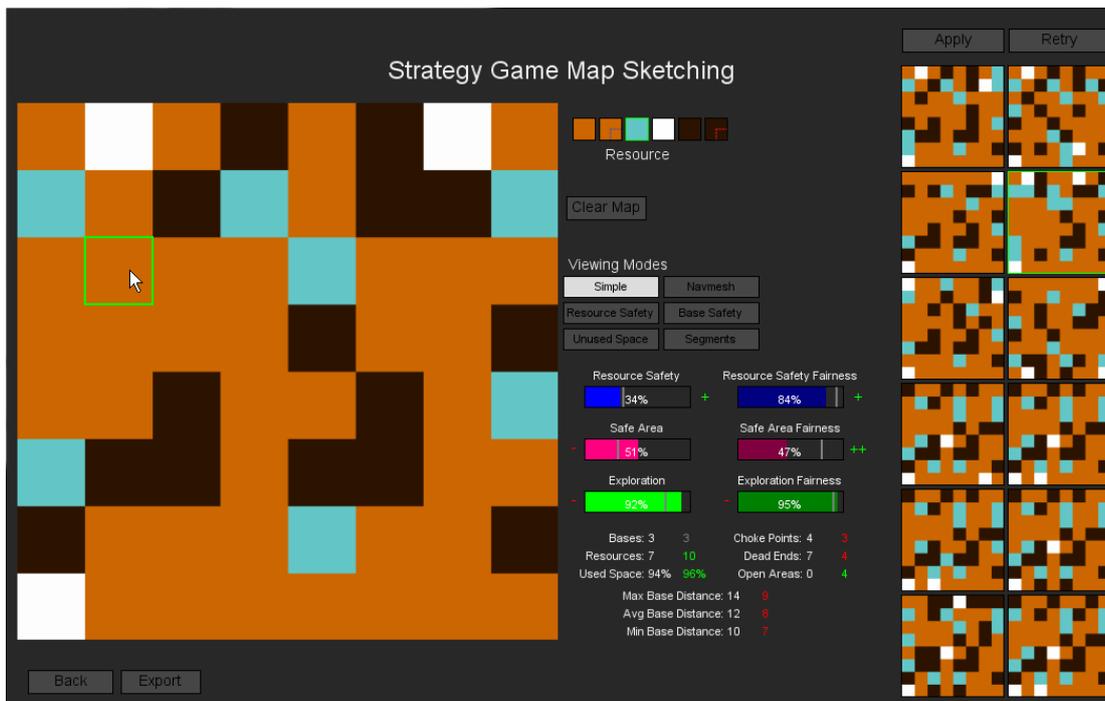
Figure j: *The Sentient Sketchbook*

4.7.6.3. Ludoscope

Ludoscope is used for progression design: the structure of missions (e.g. tasks and events) and the structure of the "space" - meaning, in a fairly broad sense, the space or world in which the player navigates within the game.

Dormans' provides a solution to the question of control and communication between designer and content generation system in the use of use of formal grammars and rewrite rules - concepts from mathematics. Rewrite rules are used by the designer as a means of specifying and shaping constraints used to generate content.

The development of *Ludoscope* is ongoing, but its current approach is moving towards Dormans' finding a solution to one of the difficult problems of game design: translating the different materials a game design into different forms: for example, how do we design the space for a mission, how do we create a mission from a given set of rules and mechanics, and vice versa. According to Dormans, we can view transitions between these forms and states as model transformations (Dormans 2012). Figure k shows a simple mission diagram modelled in Ludoscope.

69
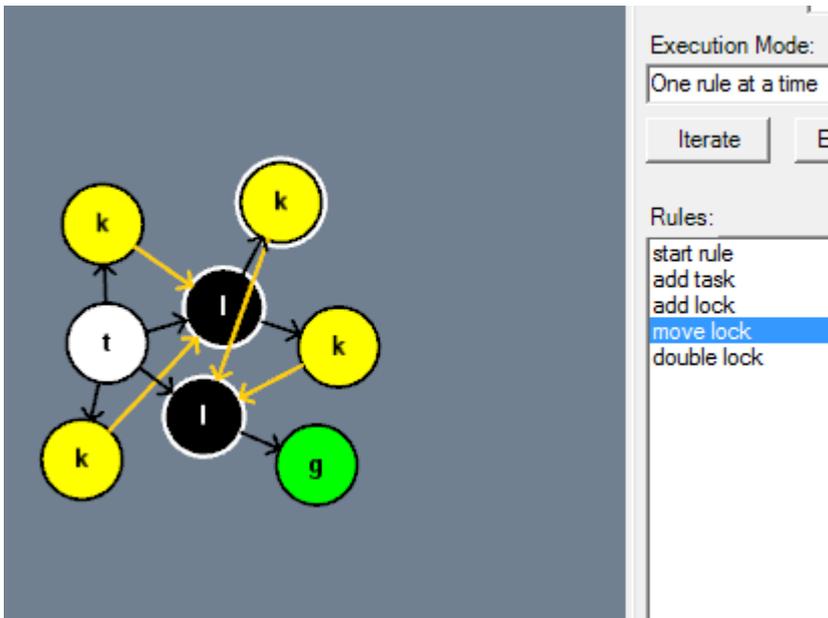
Figure k: A *Ludoscope* mission diagram

4.7.7. Progression units, rules and structures

The unit-based approach describe above has also been applied to the problem of describing game progression. For example, progression can be viewed as the acquisition of discrete skills, as in Dan Cook's "skill atoms" (Cook 2007), while some have sought to measure and quantify difficulty (McMillan 2013). Other types of units used represent goal completion (Rollings and Adams 2003) or narrative events. Unit type seems to vary as a function of not only design philosophy but game genre, some units being more useful for some games than others. For instance, mission goals for a side-scrolling platformer may be relatively simple and therefore straightforward to arrange, while skill and difficulty measures are challenging, whereas an adventure game may contain few gameplay concepts but contain a complex goal structure.

Diagramming methods have been devised to express progression in the form of rules and relationships governing the organization of these units. For example, a "tech tree" can be diagrammed for a strategy game, a "skill chain" serves as a tool for organising skill atoms, and units in the form of skill-based challenges are often organized by mapping them to desired difficulty curves (Aponte, Levieux, and Natkin 2011).

Similar concerns in terms of flow, difficulty and variety must be balanced at the macro level as well. This is the task of progression design: the design of how levels or missions fit together to

create the game's overall player progression. This structure – or "plan" – is often sketched out in a design document or on a whiteboard before levels are produced (Butler et al. 2013).

4.7.7.1. Skill atoms, skill chains

Designer Dan Cook extends the game atoms idea towards modelling the elements of a game system from the point of view of the player's experience, arguing that "to accurately describe games, we need a working psychological model of the player". A skill atom "describes how the player gains a new skill" (Cook 2007); i.e. skill atoms describe the skills a player must progressively learn in order to master the game.

These skills are essentially a different way of describing game mechanics; a skill atom in a game design is any unique action or combination of actions that the player must use to perform a given game mechanic. Pressing the "x" button to jump between platforms would count as a skill atom, for example (the game mechanic might be "platforming"). Järvinen's definitions of "game mechanics" (as player-driven operations to influence game states) versus "procedures" (as system-driven operations that respond to player operations) could be useful here (Järvinen 2008, 72). Using his definition we could say that while "game atoms" could be used to describe any game system element ("procedure"), "skill atoms" specifically engage with game mechanics.

Using diagramming, Cook suggests linking these atoms into "skill chains", tree structures that represent the order and context in which learning moments for new skills occur. Cook's intent is that such a method will help elucidate to the designer the player's core gameplay experience; that it will "help raise the level of intent and predictability in modern game design" (Cook 2007).

The goal of a skill chain is to address the task of breaking down what skills the player needs to progressively learn so they can play the game, and to decide on the order in which these skills need to be acquired. Cook suggests, and implies in the example skill chain he gives, using a skill chain for the player's first experiences with the game: the first minutes of gameplay in which the designer treats the game like sort of disguised tutorial, in which the player learns essential and basic skills such as, in the case of a classic side-scrolling platformer, for example, how to run, shoot, jump, and double jump.

4.7.7.2. Mission/Space

Progression structures can sometimes map to a player's spatial progress through a game world (e.g. a side scrolling platformer where the player is always moving forward), but where this is not or only partially the case making a distinction becomes useful. For this purpose Dormans proposes a graph-based framework ("Mission/Space") for diagramming both the environmental space and the abstract mission space in a way that allows a designer to more clearly see the relationship between the two structures, thus aiding progression design (Dormans 2010).

4.7.7.3. Articy:Draft
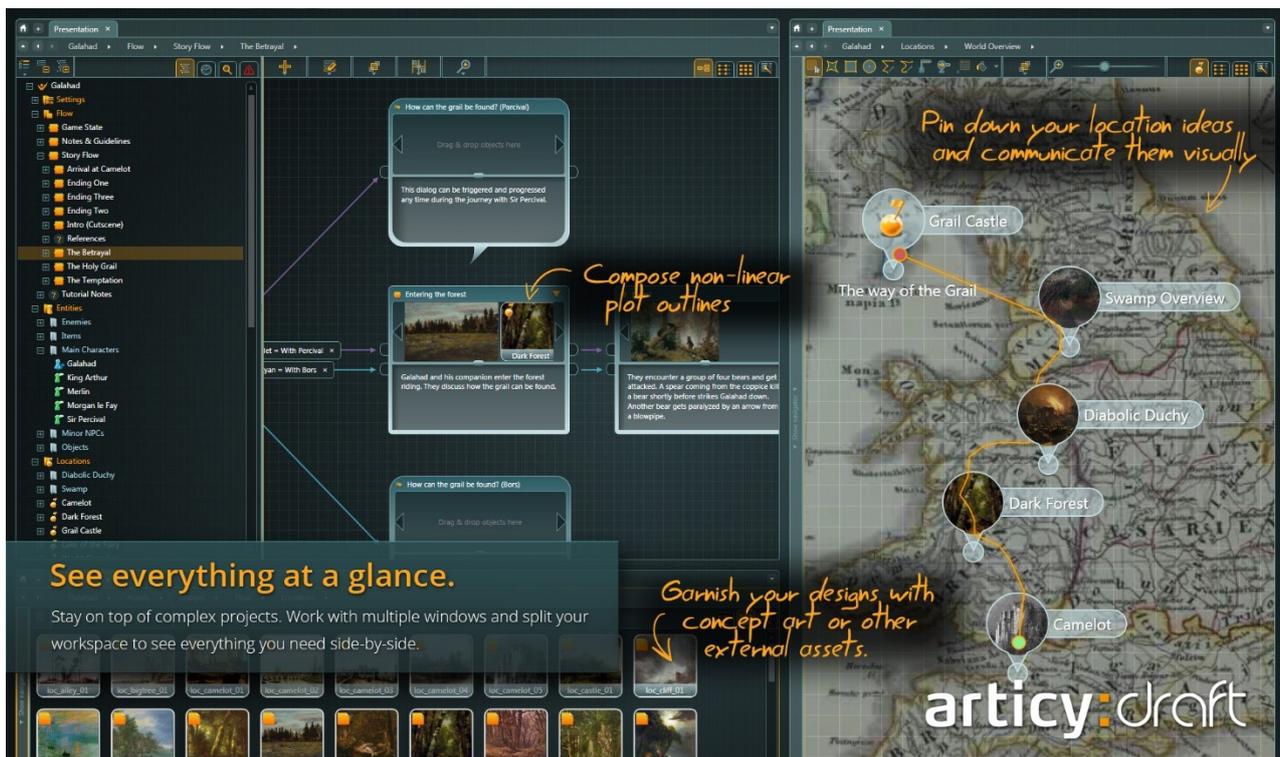
Figure l: Annotated screenshot of *Articy:Draft*

Articy:Draft (Nevigo GMBH 2011) is a commercial tool that supports design for narrative-driven or narrative-heavy games – for example role-playing games or adventure games. A flow diagram-style interface enables the design of game design elements that have branching, graph-based structures, such as branching dialogue or quests (missions). It also offers form-like graphical interfaces for the editing and organisation of design data associated with game entities such as non-player characters, items and locations. In addition, it provides an editor for creating level layouts which enable the designer to perform design-specific level editing tasks such as the placement of entities, trigger points, defining zones, and so forth. All data may be exported to be

loaded and used in the game itself (i.e. as game assets). Figure l shows a screenshot annotated by the *Articy:Draft's* developers to draw attention to some of the tool's features.

4.7.7.4. Progression planning

Butler et al have devised an approach for adding computational support to the task of progression design. This approach was implemented concretely within the context of their work on the level design tool for the research game *Refraction*. The approach is inspired by the "atomic" approach described above: the designer defines units of progression, a set of rules governing how they are introduced and used in the game ("progression constraints"), and a progression plan (an example of which can be seen in Figure m) based on these rules. The plan (which can be generated by the tool based on the progression constraints – i.e. treated as an automated planning problem) consists of a sequence of stages (levels) of the game, each stage of the plan consisting of progression units. In this way the approach not only offers the designer a tool in which to author the progression itself, but also invites the designer to externalize and impose their design logic by inviting them to define rules about the progression itself.

As *Refraction* is a puzzle game, progression units take the form of puzzle concepts (labelled "game concepts"). Butler et al have integrated these progression constraints and the progression plan components into the game's level editor (a production tool used for authoring or procedurally generating actual gameplay). Their goal in doing this is to support rapid iteration of a complete design process from planning through to playtesting and allow for the automatic detection of problems occasioned by edits to levels.

73
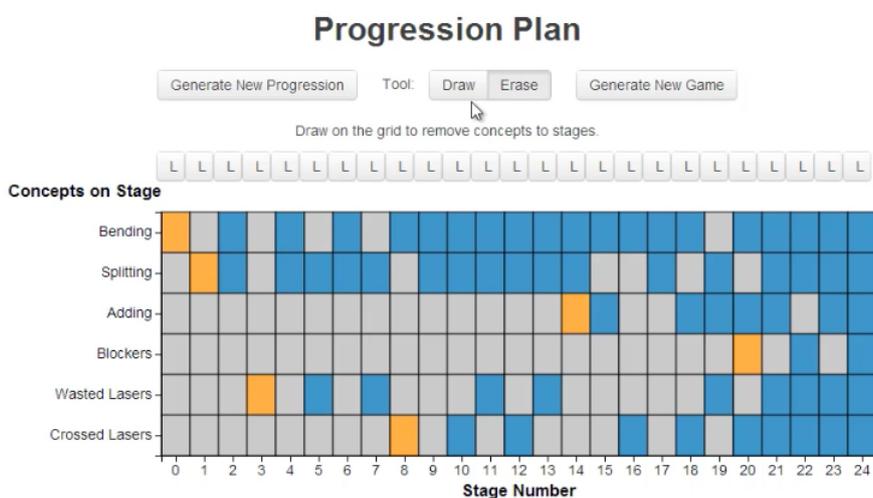
Figure m: The progression plan component of R*efraction*'s tool's progression planning interface

4.8. Supported design activities

Above I loosely classified the tools based on their methods or approaches to design problem. Here I compare the tools through the lens of design activities. Whereas above the focus was on the various approaches to the problem of design support and their technologies and implementation, here I am more taking a designer's point of view, looking at the design activities afforded by the tools.

4.8.1. Computer-aided design activities

We could say that computation in game design tools provides support in two main ways: 1) by facilitating design moves; 2) by predicting and describing game dynamics.

4.8.1.1. Facilitating design moves

First, and fundamentally, the software offers an enhanced interface for representing the design situation and viewing it.

The software facilitates the making of design moves made directly but indirectly (mixed initiative design tools like *Ludoscope*, and *The Sentient Sketchbook* transform the designer's high level design moves into concrete gameplay, for example).

This can include constraining and informing design moves that the player makes, based on rules they have defined either directly or indirectly (in the case of *Refraction*, progression rules are computationally derived from rules specified by the designer).

4.8.1.2. Predicting and describing game dynamics

Second, computation can predict and describing game dynamics. In other words, to address the "second order" problem of game design (which, as we recall from in Chapter 3, is the fact that the design of a game only indirectly affects the player's experience of it) in response to the hope, expressed by many, that this can help us avoid some common design pitfalls (Hunicke, Leblanc, and Zubek 2004).

The greater challenge is in predicting and describing game dynamics. This is because gameplay – in other words, the behaviour, or dynamics, of a game – is produced by combining ingredients: a

game and a player. As we know, prototyping a game and playing it (and by extension, analysis of player metrics) is currently the gold standard (arguably, the only standard) for predicting and describing game dynamics. By designing a game in a tool, the tool has access to the first ingredient: a model of a game. For the second ingredient, however, the tool needs an alternative to a human player.

One solution, as Nelson argues, is that AI techniques afford strategies for extracting information about the dynamics of a game from the game artefact itself. Instead of plotting the experience of individual players, the tool takes the point of view of all possible player decisions, finding the thresholds of what is possible in the game, what is impossible, what is necessary in order for given play outcomes or scenarios to be occur, and so on (Nelson 2011). Smith, for example, talks about his tool *Ludocore* as a system that "imagines gameplay". It offers a representation of the game that includes this imagined gameplay – in the form of a "vast space of potential low-level action sequences possible in a game that satisfies arbitrary logical constraints".

Another solution is to simulate the game using the designer's model of the game as specifications, and then either 1) have the design interact with the game simulation (e.g. *Machinations*); or 2) simulate a player to interact with it i.e. as a kind of automated playtest (*Ludocore*). To simulate a player, the tool needs a player model – this is described further on in this chapter.

4.8.2. Two forms of representation: model and dynamics

Stepping back to consider all game design tools, whether or not they offer computational support, I want to now reconsider the distinction I made above through another lens: the representation of the design situation. Through this lens one could describe game design tools as supporting two forms or "orders" of representation:

1) A representation of the game artefact (the game mechanics or progression structure, for example). This is the *model* of the game, created via the design activity of "representing" – i.e. directly by the designer.

2) An additional form of representation is offered by game design tools that offer computational support for describing *dynamics*. One might say that the tool is, in design terminology, doing some "representing" of its own. To do this it takes the designer's representation – their model – as

formal specifications, and adds to it the second ingredient of the "player" (in the form of a procedural playtest, AI algorithms, or the designer's interaction with the simulation).

The goal of this second representation (the dynamics) is to bridge the gap between the rules of play (the model) and the play experience. In a sense the tool is bringing the "second order" design problem into lower-order view for the designer. This representation or view might be a real-time simulation or it could be in some more arcane form. For example, in *Ludocore* this representation would be viewed through the lens of symbolic gameplay traces that respond to queries from the designer.

It is in this way – the two forms of representation - that the natural comparison between computer-aided design tools for game design with computer aided design tools for architecture and visual design begins to fail us. Conventional architecture, for instance, does not have a "second order" problem to solve. Computational support offers game designers more than just a way to create static models – like architectural CAD software; it also offers the opportunity to simulate system dynamics from those models. For comparison we can instead look to other kinds of system designers who use design tools: software engineers and computer scientists[15].

Like several of the game design tools reviewed earlier in this chapter, some modelling languages for software design use graphical notation to describe the structure and behaviour of systems. UML is the well-known example of one such language. As we can see earlier in this chapter, both UML and Petri-nets, a mathematical modelling language, have inspired several proposed game design modelling languages.

A key difference between languages is in whether they are formal, semi-formal or informal. Informal (e.g. natural language) and semi-formal (e.g. graphical) notation does not have its syntax nor its semantics completely defined. As such, its models can be ambiguous. Whereas, formal notation is unambiguous with completely defined syntax, and hence free of ambiguity. This means that models built with formal specification languages can be read and understood by computers, allowing them to infer useful information, and serve as specifications for performing analysis and verification of a design, as well as creating simulations.

---

[15] It is important to remember though that game designers, as distinct from game programmers do not design any form of software system – they design game systems. i.e. in the sense that Chess, for instance, is a system.

The disadvantage of formal specifications, however, is that these languages are more difficult to learn and harder to understand by humans. Models represented using semi-formal specification languages, therefore, are ideal for the purpose of being read and analysed by the designer – the designer understand the *model* of the system, in its static form. Formal specifications, on the other hand, because they can be read and analysed by a computer, allow the designer to understand the *dynamics* of the system they have designed as mediated by the computer.

We can see this same difference in game design tools: between some of the diagramming methods that stand alone, and those that are supported with computational features that offer insight into game dynamics. The pen-and-paper game diagramming languages that do not provide computational features provide one kind of representation: the model itself. Hence they are (or at least, they probably should be, for design purposes) semi-formal, focussing on being read and understood exclusively by a human designer. *Machinations*, on the other hand, must use a formal notation language (*Machinations'* language is inspired by Petri-nets, a formal specification language) because it provides, in effect, not one but two forms of representation of the design situation: i.e. the *model* and the *dynamics*. The first, the model of the mechanics, should ideally be as readable and useful to the designer as possible but, most importantly, it must be understood by a computer, in order that it can provide the second kind of representation: i.e. the *dynamics*.

4.8.3. Player modelling

A player model is an abstracted description of a player or a player's behaviour in a game (Dormans and Bakkes 2011). Player models are used in adaptive gameplay (where the game adapts to individual players' play styles), but are also used game design support.

As well as contributing to the task of simulating gameplay (as discussed above) by affording "procedural playtesting" (Holmgard et al. 2014), it offers designers a way to understand how that gameplay would vary as a function of different play styles. Nelson et al., when interviewing designers about what they might like from a design tool report interest in being able to model human play styles:

> One large category of design questions they had was what gameplay would be like for different types of players; for example, how would the player fare who always picks up the strongest armor and weapons they can find, uses health potions, and does nothing else? (Nelson and Mateas 2009)

Ludocore provides an interface that enables the designer to create player models. *Machinations* does this with its "Artificial Player" node type – simulates interacting with the diagram in accordance with scripted logic written by the designer. It controls interactive nodes in the diagram[16].

4.8.4. Producing and transforming design materials

Referencing HCI research, Khaled et al suggest the applicability to game design tools of a St Armant and Horton's distinction between "effective tools" and "instruments" (Khaled, Nelson, and Barr 2013). While tools St Armant and Horton class as "instruments" afford the designer information about their design, "effective tools" produce a persistent effect. "Effective tools", in other words, are tools that produce and transform design materials (St Amant & Horton, 2002).

For example, Gillian Smith's mixed-media level design tool *Tanagra* can be used as both an "effective tool", allowing users to edit a game level, and as an "instrument", providing visual feedback if the user makes a change that leads to an unplayable level (Khaled, Nelson, and Barr 2013).

With tools that feature procedural content generation, the production of design materials comes to the fore. These tools (for which Khaled et al use the metaphor "materials"):

> …[afford] choice and abundance to the designer. The designer's task then can shift towards choosing, selecting and tweaking materials, rather than developing them manually in their entirety (Khaled, Nelson, and Barr 2013)

But a persistent change to the design materials could take the form of any persistent change to the design situation, for example modelling a system (*Machinations*, *Ludocore*), generating or adjusting game content based on constraints expressed by the designer (*Tanagra*, *Ludoscope*), or even transforming the constraints themselves (progression planning).

4.8.5. Explicit computer-aided back talk

As previously discussed, the discovery and diagnosis of design problems is a function that designers have expressed particular interest in, in terms of what they would like from a game design tool. This raises the question of how a tool helps a designer discover design problems.

---

[16] See http://www.jorisdormans.nl/machinations/wiki/index.php?title=Artificial_Player

To find problems, a designer can read and analyse a representation of the design situation. One of the essential characteristics of design problems, however, is that they are often not apparent and must be found (Lawson 2006, 3rd revise:56). In this context, asking a computer to find problems on the designer's behalf is useful. It is considered that computational media in particular can help designers to identify breakdowns that they may not be aware of (Fischer 2004).

Koster, for example, suggests the idea of tools allowing us to detect degenerate strategies [17] in a game design (Koster 2005). Cook talks about the need for us to develop a model of game mechanics that is "testable". When interviewed, designers told Nelson and Mateas that they would be interested in a tool that they could use to answer questions about their designs:

> Do all objects (units, buildings, etc.) play a useful role? Given the interactions between the game subsystems (economy, base defense, etc.), do the gameplay dynamics avoid overly convergent, dominant strategies (Nelson and Mateas 2009)?

As discussed above, *Ludocore* provides a query interface: it allows the designer to ask specific questions, and get specific answers.

Another way to detect problems in a design that researchers have used or proposed using is automated testing based on approaches used in software engineering. Araujo and Roque, for example, propose that design tools that provide verification, validation could help find balancing issues or problems within the game's flow (Araújo and Roque 2009). In software engineering, validation and verification are tests performed to see whether the software specification captures the client's needs and whether the software meets the specification.

Another idea borrowed from software engineering is regression testing. This is a kind of testing specifically performed after changes to the system are made, in order to detect new bugs ("regressions") arising from these changes. Nelson suggests a form of regression testing could be used in game design tools:

> If design goals identified in the exploratory phase are noted, then during the testing phase, a series of regression tests can be run to make sure the design goals haven't been broken by recent changes, much as in software engineering regression tests check to see if previous bugs were reopened by new modifications (Nelson and Mateas 2009).

---

[17] Degenerate strategies are strategies players can use to exploit defects in the game system to play in ways the designer does not intend.

Tanagra performs this function (albeit for a level design rather than a game system model) – as noted above, it provides visual feedback to alert the designer if they make a change that leads to an unplayable level.

Finally, though not strictly finding problems, the constraining of design moves and/or the automatic adjustment and refinement of them by the tool as or just after they are made helps ensure that (some kinds of) design problems do not occur in the first place. This seems to be a traditional Computer-Aided Design approach. One of the key features of *SketchPad* (Sutherland 1964), a very early computer-aided design tool, was enabling the designer to constrain geometric properties such as line length and angle size. Checking design moves against constraints features in *Tanagra* and *Refraction*'s tool.

4.8.6. Evaluation using interactive simulation

In current game design practice, the dominant evaluation activity is play-testing. This fuels a need to start prototyping and producing as soon as possible. Playtesting, however, is resource-hungry: not only does it require building a playable game prototype, it can be very time consuming; some questions about our designs may only be answered through data collected from hundreds of play sessions.

Therefore this is a desired role of game design tools – that we can evaluate designs that are now commonly only testable via playable prototypes.

Above I described how computer-aided game design tools offer simulation as a means of extending the representation of the design situation. Such tools can be interactive, offering the designer the ability to interact with the simulation, in a manner similar to "playing" the game. Notably, *Machinations* and *BIPED* have this functionality.

4.8.7. Communication features

The separation of production from design requires that communication become one of the key tasks of a designer. As Lawson notes:

> If the designer is no longer a craftsman actually making the object, then he or she must instead communicate instructions to those who will make it (Lawson 2006)

On AAA development teams, communication of design specifications to the team has long been a major part of the designer's role. Communicating specifications to programmers and artists is

important. While current practice in this area could no doubt be improved upon, I have not experienced this to be a major challenge in the past on conventional projects, using existing communication methods (technical documentation, verbal discussions, flow charts, etc.), nor have I seen it expressed as a difficult problem that designers want resolved in order to improve design outcomes.

While game programmers frequently themselves have to "fill in the gaps" for where detail in game design communication is missing, I would argue that this is not a communication problem. Instead – based on my experience on both sides of the programming team/design team communication loop - I attribute it to the lack of design tools; i.e. the design thinking itself has not taken place, therefore the design situation lacks detail. Instead, it is the game programmers who have the game in which to model and test details of design that are difficult to model and evaluate without computational support (AI for non-player characters, for example), and so the programmer might be effectively taking something of a design role – i.e. they are "designing-by-making".

Being able to codify design knowledge in order to communicate with other designers, however, has been identified as a problem for designers. Church characterises our lack of a means to communicate our ideas effectively between designers based on a "common design vocabulary" as "the primary inhibitor of design evolution". Understanding, he says, requires that designers be able to communicate precisely and effectively with one another (Church 1999). This kind of communication is very different to implementation specifications, however; as the information is not designed to be used to produce a game from the design but rather to afford understanding of the design, the language of design does not need to be in a form or level of abstraction that is directly transformed into gameplay. Indeed, design knowledge can take a form that is even slightly philosophical (take Jesse Schell's *The Art of Game Design* (Schell 2008), or Steve Swink's *Game Feel* (Swink 2009), for example – both books written to communicate design knowledge from designer to designers).

While this is a valuable question, working as the sole designer on my case studies I am not able to evaluate this attribute of designer-to-designer communication in practice, and so I will not be addressing it in this study. My focus is rather on tools that can support design thinking - the "communication" that occurs within the design conversation with the materials.

4.8.8. Comparing the tools

Table 1 below summarises a comparison of game design tools in relation to the design activities they support, as discussed above. In addition, using a domain-specific lens, it indicates the types of design tasks they ostensibly support. By tasks I mean not so much the type of design activities performed as the parts of the game design situation being worked on. These were discussed in Chapter 2, where I described contemporary game design roles and their associated tasks: concept development (ideation), progression design, gameplay authoring (level/mission design), system design and balancing.

Table 1: Comparison of design activities and tasks in game design tools

	Design activities									Game design tasks			
	Generating materials (game content)	Producing or transforming materials (persistent change)	Information-giving	Constraints checking or validation	Explicit back talk	Simulation	Interactive simulation	Communication	Exploration focus	Concept development (ideation)	Progression design	Level/mission design	System design and balancing
Machinations		x	x			x	x	x	x	x	x		x
Progression planning (within Refraction editing tool)		x	x	x				x			x		
Sentient Sketchbook	x			x								x	
Ludoscope	x	x							x			x	
Game design patterns										x			x
Skill chains		x										x	
Tanagra	x				x							x	
Sketch-it-up!							x		x	x			
Ludocore + BIPED					x	x	x					x	x
Articy:Draft	x	x						x			x	x	

5.   RESEARCH QUESTIONS AND METHOD

5.1.   Overview

This chapter outlines the motivation for this project and the contribution it makes to the field of research on tools for game design. It presents my research questions and lists the tools included in this study, as well as the research method used and the thinking behind it.

5.2.   The need for evaluation

The involvement of respected practising game designers in this push towards developing models and formal approaches to game design suggests that designers themselves would, in theory, find such approaches useful to support their practice. But this should not be automatically assumed. It is important to recognise who these game designers are and who they represent: an elite minority of the game design community who meet and discuss their ideas regularly at events such as the industry's international Game Developers Conference (GDC) but also at exclusive invitation-only events such as "Project Horseshoe", an annual gathering of game designers dedicated to solving current problems in game design. Just as 'best practice' methods like paper prototyping may appear frequently in game design textbooks but far less frequently in the workplace, the design ideas and methods described by an elite minority of game designers should not be taken as representative of the typical methods used by the average professional game designer.

It is hard to measure the influence of frameworks suggested by these game designers and researchers upon practising game designers. Certainly in my own experience, while I have spoken to some designers who are aware of these theories I am not aware of any of my peers who use them in their practice. Even Koster admits that he "can't yet picture designing a game" with his own notation system (Koster and Ebrary 2005). The exception is perhaps the Mechanics, Dynamics Aesthetics (MDA) theory of game design, which forms the basis of the Game Design Workshop at the annual Game Developers' Conference in San Francisco. But no formal methods based on this analytical framework have been published to date, nor any accounts of designers using the MDA framework in their practice.

If techniques proposed by these vanguard designers – such as Raph Koster with his "game grammar" and Dan Cook with "skill atoms"- are not filtering through into common design practice,

we have also to consider why. One explanation could be that the ideas of the elite are more advanced than those of the majority, but we must also consider other reasons. It may be, for example, that the experimental, risk-taking concept work of creative directors like Koster calls for quite different design methods to the everyday craft work undertaken by a typical game designer. In 2008, Project Horseshoe published a report in which they appeared to consider this possibility, worrying over questions such as "are skill atoms a pragmatic tool for working designers?" and expressing a concern that although "in recent years a new set of tools for building games has emerged", they still remain hobbled by the problem that "most descriptions are highly theoretical and only considered useful to pointy headed academics or their mad inventors"(Blinn et al. 2008).

Dormans, while observing that none of the attempts to introduce formal models for game design has been so successful that they have become an industry or academic standard, attributes this to the fact that they "tend either to be too mathematical for the diverse population of game designers and scholars, or were not explored or presented with enough detail". He adds that, most importantly, they require designers to make an investment by learning a new paradigm – an investment unlikely to be made unless there is an "obvious return" in the form of a better, more efficient design process (Dormans 2009). This sentiment resonates with me as a practitioner. As a game designer, my first instinct is to ask: will design support tools and methods aid our design work? If so, which tools work best for which of my design tasks? From a practitioner's perspective, evidence of how these theorised tools and methods fare when applied to real game design problems is crucial.

Some argue that the design models proposed are as yet too underdeveloped to be used by designers. Designer Stephane Bura, who proposes a solution to what he describes as the "game diagramming problem", qualifies his contribution as an early attempt that needs further work before it can be of practical use. "Barring [additional work]," says Bura, "this grammar will remain a simple descriptive tool instead of an analytical or even a design tool" (Bura 2006).

A similar observation could be made about tools developed in a research context. Some of these tools, though they have been built and tested, are early stage work and not intended to be for general consumption. Their role is to inform the development of future tools that might be created to be used in the field. Adam Smith explains that such tools were created to function as "computational caricatures", exaggerating and oversimplifying their features in order to make their technical claims about the potential role of AI in the game design process clear and easily

recognisable. He argues that systems like *Tanagra* and his own *Ludocore* are not intended to be used as is, in the field, but instead have been built to offer "a tangible, interactive demonstration of a potential future for level design tools and a validation of the software architecture that made it possible".

So do we know yet whether there is an "obvious return" on investment to be had from integrating design tools into a game designer's practice? In general terms: no, not really. At least one influential designer has expressed doubts as to the potential usefulness of formalisation and abstraction for game design (Schell, 2008: 145), and thus far we can furnish little evidence to counter such scepticism. To date we have little data upon which to base any such assertion, let alone if any one approach to this problem is more fruitful, in practice, than any other. In all, we have little upon which to base real insight into these tools from a practitioner's point of view, let alone enough empirical support to help resolve the question of whether formalising the game design process with tools and formal models really would aid or improve game design practice.

This is because we lack sufficient data of the kind that practitioners typically find most meaningful: published accounts and analyses of designers' experiences. Game designers – if they are using these tools – are not conducting their own evaluation research; which, in the case of the game design community, means post-mortems shared via conferences or trade journals. Researchers, on their side, have not yet conducted any deep and thorough-going evaluation research of experimental design techniques and tools. Dormans has said in relation to his *Machinations* design tool that "it remains to be determined how easy it is for designers" (Dormans 2011). And while the developers of *Sketch-It-Up!*, a tool developed until 2009 at Carnegie Mellon University, workshopped their tool with children, there is no discussion of how the tool fares in the professional design contexts it was built to support (Karakaya et al. 2009).

The result of this is that, despite the fact that research in the area of game design support has been emerging for several years, to date there has been relatively little discussion or comparative critique of this work. Even at a purely theoretical level, the field lacks work that provides comprehensive comparative analysis that is based on real practice. Such comparative analysis would be useful: while all these proposed design support solutions tend to share core foundational elements, their approaches and results differ. But this absence of analysis is not surprising – indeed, it would be hard to develop an honest critique of game design tools in the

absence of practice-led research. Without being able to learn lessons from previous work it is hard to justify taking new directions with new work.

Then again, viewed within the context of academic research, this situation may not seem unusual. In Computer Science research and its related disciplines we prefer projects that develop novel technology and advanced techniques. Evaluation in this context has a support role; its purpose is to legitimate and validate this activity. As such, evaluation in Computer Science and related fields is typically not a research contribution – it validates research contributions. The kind of information we need about practice is not this kind of research validation - a kind of prompt, efficient, anaesthetized outpatient medical procedure. Rather, the introduction of tools into a crafts-based design practice calls for an evaluation process more akin to an emergency room organ transplant: traumatic, messy, long and difficult.

5.3.   This contribution

We need evaluation of this work that goes beyond simple research validation. Accordingly, this project aims to contribute to our understanding of the work on this question from the perspective of design practice.

This project aims to contribute knowledge that can help inform future advances in this emerging field of game design support. Taking the form of a designerly line of enquiry into this question, it comprises a longitudinal, practice-led evaluation research conducted as a participant-observer, applying game design tools to longitudinal game design case studies in order to generate observations that can feed back into further development in this area. My observations, analysed through the lens of what we know about design support, have revealed where some problems and potential of a tool-supported game design practice might lie.

As a secondary goal for this project, this work aims to offer material that can contribute towards filling the literature gap vis-à-vis work on game design as a design activity and a practice. As noted above, there is a scarcity of published material in this area. As a research community it is too easy to find ourselves bedazzled by the ever-changing technical challenges and new possibilities for game design, while forgetting that we still know relatively little about game design itself.

5.3.1. Research questions

Designer and developer Chris Hecker once observed that the transfer of ideas between academic games research and professional development is rare. It is surprisingly hard to develop a new idea that can demonstrably and meaningfully improve game development practice in the field. He counselled researchers (those who aim to propose ideas that can be used in practice) to keep their offerings unambitious and simple. "All we want is two sticks and a rock" (Hecker 2011). There, Hecker was speaking about technology for game production. Researchers who develop tools and technology for game design face an even harder challenge. They are not just proposing the addition of this new modelling language, or that new interface – they are proposing serious disruption to the status quo. As I argue in this thesis, game design is currently still a crafts-based practice; as such, design tools pose an aggressive, paradigmatic challenge to our current approach to game design at a very basic level, in that they propose a means of separating design thinking from making. In turn, this may even represent a challenge to the identity of game designers themselves, in terms of the way they see themselves and what they do.

Other researchers, by creating game design tools have proposed a future for game design; my focus is on thinking about what the first step towards such a future might be. This wide gap between the advanced, sophisticated design technologies being explored in academic research and the (often, quite literally) paper technology of real-world game design practice must colour the nature and approach of any evaluation of design tools. It is a context in which the success of a very first step – however minor or primitive – would be significant. In essence, to borrow Hecker's metaphor, it means asking: where are the two sticks and a rock that game designers might one day use to light the first fire?

With this general approach in mind, I had three main research questions to direct my enquiry.

Firstly, can game design tools improve game design practice? One theme expressed by advocates for game design tools concerns improving the quality and efficiency of the game design process. Tools might give designers the kind of insight into the emergent properties of their game mechanics (Koster 2005; Leblanc 2005) and lessen reliance on the relatively costly process of prototyping them; tools might help designers better articulate and communicate their ideas, with their colleagues, but also with each other, allowing design knowledge to be effectively codified and shared (Church 1999).

Secondly, can game design tools expand the scope of game design outcomes? In other words, can game design tools enlarge the possibility space of game design? Another theme expressed by tool advocates is concerned with advancing, or expanding the nature of game outcomes. Designers (Church 1999; Cousins 2004b; Cook 2007) have articulated a sense that a lack of design tools is holding us back from pushing creative boundaries and moving design forward. One of Dormans' stated aims for his tool work, for example, is to provide a way of reconciling the design of game mechanics with the design of game progression.

Thirdly, how do design tools change or impact the game design process? Several design tool researchers have observed that the use of design tools strongly affect designers' courses of actions and thought processes (Resnick, Myers, and Nakakoji 2005). This is perhaps the most interesting question, given the radical nature of introducing formalisation to a relatively informal design practice.

Finally, I hoped to discover knowledge that could contribute to future game design tool development research – by revealing some of the problems, successes, and possibilities of game design tools based on my experiences.

5.4.   Tools used in this study

As discussed previously, the game design literature reveals that "brainstorming solutions to problems" is already the go-to design tool (Kuittinen and Holopainen 2009) in the absence of other tools to support design thinking. While this activity could perhaps benefit from support, I did not have the sense that reaffirming the worth (or not) of card-based brainstorming aids would be a useful contribution towards the work in this field. A strength of tools of this kind is that, given their focus on the ideation stage of game design, they are comparatively straightforward to evaluate; indeed, typically the card designers have been able to test their card systems quite extensively and show that they can be useful. It would be surprising though if they were not, as other design fields have already confirmed the worth of card decks for brainstorming and creative brainstorming; as such, while modular concepts and pattern languages for game design are relevant, tools in the form of sets of cards are not especially pertinent to the problem of tools specifically for game design.

Aside from this, my selection criteria were rather pragmatic, in that I tried to include every game design tool available to me. This was due to the fact that there have been, to date, relatively few attempts at creating game design tools. My choice of tools to use in this study was a function of practical constraints symptomatic of how early and how fringe work in this research domain is.

Some tools are simply unavailable - *Sketch-It-Up!* (Karakaya et al. 2009), for example. Where software tools have been created they are often not advanced enough in the work to create a tool that can be used by a third party; rather, they are research projects rather than tools designed for wider use. They have been designed for and validated within the context of a specific game and development context created by the researchers themselves. This is the case, for example, with *Refraction*'s tool and *BIPED/Ludocore*. Where they can be used by a third party, they have been created primarily for the purpose of validating the research and can only feasibly be used on the game that was used as a case study. Despite my attempt to be inclusive as possible, therefore, practical constraints have meant my selection could not be said to be an ideally representative sample.

System modelling

- *Machinations* (Dormans 2009)

Narrative design

- *Articy:Draft* (Nevigo GMBH 2011)

Mixed-initiative design

- *Ludoscope* (Dormans 2012)
- *The Sentient Sketchbook* (Liapis, Yannakakis, and Togelius 2013)

Progression design

- Skill chains (Cook 2007)
- Progression planning (in *Refraction*'s tool) (Butler et al. 2013)

5.5. Contributing a designerly line of evidence

What is the correct method for evaluating game design tools? It is here, where research into the design of games is poorly served by its disciplinary context.

Unlike some of its counterparts, such as Music, Art, Film and Theatre, research into game design and development practice does not yet have a home of its own. Despite over a decade of "Games Studies", games research still bears the legacy and methods of the Humanities and Computing disciplines it emerged from. While player experience research dominates the Humanities-led Game Studies, much of the practical work relating to the design and production of games themselves tends to find a home within the technical disciplines of Computer Science and HCI. Hence topics that, in parallel subdomains (e.g. Computer Music) might be considered to be art or design research topics, are contextualised within science and approached using the methods of scientific research. This despite the fact that, given what we know from at the other creative disciplines within academia, it seems probable that game design research could benefit from a wider range of methods of inquiry – particularly designerly methods.

The literature on the evaluation of design tools, particularly design tools for the creative disciplines, is not extensive. The material I use here for guidance on evaluation methods has largely come from Creativity Support Tool evaluation within HCI research, Art and Design, and Design Studies.

The Creativity Support literature suggests that there is no one correct method. Not only is there a variety of valid methods available for approaching this question, but a diverse methods of enquiry should be used in order to produce "converging lines of evidence". Terry, Mynatt, Nakakoji & Yamamoto point to the importance of employing mixed methods for evaluation, arguing that a "richer suite of evaluation instruments" is necessary for the study of tools (Terry, Mynatt, Nakakoji & Yamamoto 2004). This is echoed by participants of the *National Science Foundation Workshop on Creativity Support Tools*, who shared the view that multiple evaluation methods are required for assessing the different aspects of creative work and that a "FAMILY [their emphasis] of evaluation techniques which must be brought to bear in order to CONVERGE on the key issues involved" (Ben Shneiderman et al. 2005).

One reason given to have multiple evaluation methods for design is to compensate for the fact that any design process evaluation method is considered to be inherently flawed. Indeed, Lawson

asserts that "any one experiment on the nature of the design process is likely to be flawed in some way" (Lawson 2006, 3rd revise:41). Owing to these common inherent flaws, any single research methodology may be "inappropriate as a single tool for a thorough, meaningful interpretation of a result" (Ben Shneiderman et al. 2005).

Similar sentiments regarding method have been expressed in the literature on Art and Design research methods. In their *Guide to Research Process in Art and Design*, Gray and Malins suggest an 'artistic methodology' for research in art and design that cautions the creative researcher against being excessively narrow or rigorous in their choice or application of creative method:

> Characteristics of 'artistic' methodology are a pluralist approach and the use of a multi-method technique, tailored to the individual project. Methodology should be responsive, driven by the requirements of practice and the creative dynamic of the art/design work. It is essentially qualitative, naturalistic and reflective. […] It also demonstrates a willingness to examine other fields and make sensible connections. It requires an outward-looking attitude and an awareness of other research cultures and paradigms. (Gray and Malins 2004)

5.6. A method for studying design tools

Here I describe an evaluation method appropriate to this work. As a practitioner-researcher studying tools for design, a designerly form of enquiry seemed appropriate. It is not the best or the only possible method but it is one that suited the parameters of my project and areas of expertise. My goal was not to provide definitive answers but to complement other studies, offering a single piece of the puzzle, or a single line among "converging lines of evidence".

To summarise my method: my inquiry is practice-led, using a method of participant observation as a reflective practitioner. I used a range of design tools within the context of design work on a group of longitudinal game design case studies. I also used conventional design workflows and tools (prototyping, documenting, etc.) where they seemed most appropriate, adapting them where necessary.

The case studies provided a meaningful context within which to apply game design models and tools to 'real-world' design problems, from which I was able to develop a practical understanding of what it is (or might be) to design games with design tools. I used a design diary to record my thoughts and experiences, which formed the basis of my observations (presented in Chapter 7). The observations of my experiences with the tools were analysed (Chapter 8) using knowledge derived and adapted from design support and creativity tool literature.

5.6.1. Evaluation, not validation

Evaluation is considered an important part of the research process in Computer Science and HCI. At a panel session on evaluation at the Procedural Content Generation Workshop at Foundations of Digital Games in 2013, panellists expressed the view that insufficient evaluation was being carried out in the field. It was suggested that insufficient time and resources were being allocated by researchers to this task for the evaluation of their own work, and that it was hard for reviewers to assess the quality of conference submissions on work that did not show adequate evidence of validation of a high standard. This feeling is not uncommon. A lack of evaluation has been the subject of debate within the HCI domain. Ellis and Dix, based on their analysis of 170 *Infovis* papers, highlighted insufficient evaluation (by researchers of their own work) overall, as well as what they considered to be limitations and serious problems (e.g. studies with foregone conclusions, wrong sort of experiment, fishing for results) (Bertini and Plaisant 2006).

The prevailing expectation here is that evaluation is the responsibility of the researchers who created the tool or technology to be evaluated. Indeed, evaluation conducted by researchers in this domain is typically for the specific purpose of validating their research contribution (or else face the prospect of being unable to publish). While this form of evaluation is expedient in pragmatic terms, its narrowness of scope is not necessarily fit for purpose in terms of answering deeper research questions.

Some are even of the opinion that having researchers evaluate their own work has detrimental effects on the quality of research. Lieberman, addressing his fellow HCI research peers, calls the scientific integrity of this convention into question. He asks us to consider that medical researchers, for example, "would be horrified at the idea of someone who developed a technique being the one to evaluate it", as studies by people originally involved with whatever is under study have experimentally been shown to be biased (Lieberman 2006). Meanwhile, in the Creativity Tool Support domain, Shneiderman and Plaisant discuss what it means when the motivation for tool evaluation is driven by the researcher's own need to validate their research contribution. They claim that the evaluation aims become narrow, the researcher's overarching goal being to refine their tool and claim enough success to warrant academic recognition (towards the publication of a paper, for example) or further commercial development. This narrows the focus to the goal of demonstrating the "goodness" of a proposed technique, creating a tension between that and any reporting about its potential or observed limitations (Bertini and Plaisant 2006).

It could be useful, therefore, to see evaluation for the purposes of validating a research contribution (and within the context of that research contribution) – i.e. validation – as its own particular and necessarily narrow type of evaluation. In contrast to the needs of validation, my research inquiry required a broad and open approach, conducive to discovery. At the very least, a contrasting method can offer a different direction or line of enquiry to contribute to converging lines of evidence.

Evaluation oriented towards discovery runs counter to some conventional HCI evaluation approaches, where evaluation is limited to and biased towards criteria and effects that can be measured and quantified within the researchers' methodological constraints (Lieberman 2006). The problem with this is, according to Schneiderman and Seo, that knowing what all the evaluation criteria or research questions should be may not even be possible before the experiment evaluation of design tools. Some questions are "discovered" during the course of evaluation, making the process "somewhat exploratory and time-consuming" (Seo and Shneiderman 2006). Shneiderman and Plaisant recommend the researcher evaluating creativity support tools remains flexible; that they be open to adapting the goals and the process used during the evaluation study itself (B Shneiderman and Plaisant 2006).

Even specific evaluation goals – for example, goals based on the stated goals of the tools themselves, may be too limiting. This thinking can also be seen in some design research methods used in the social sciences. 'Goal-free evaluation' (Scriven), is an approach that relies heavily on description and direct experience (Patton 1987: 36). It is premised on the idea an evaluation should examine value by investigating what it is actually doing rather than what it is trying or claiming to do.

As my investigation is into relatively uncharted territory, the adoption of an open, exploratory approach seems all the more relevant. Concretely, I might find that a tool I thought (and the tool developer thought) would be useful for a particular design task for a particular game genre is proving to be ineffective, but I might discover that using it a different way, and with the 'wrong' set of criteria and expectations, proves fruitful.

5.6.2. Reflective practitioner as participant-observer

In evaluation research there are two main methodological approaches: the hypothetical-deductive model used in science; and "naturalistic enquiry", a qualitative approach that relies upon the

interpretative skills of the observer (Somekh and Lewin 2005). Longitudinal participant observation is becoming increasingly popular as a research method in creativity support tool evaluation, according to Sneiderman and Plaisant, who observe a trend away from usability studies and controlled experiments. (B Shneiderman and Plaisant 2006). Similar trends have been noted in Art and Design, where there has been a shift from "what could be seen as 'positivist' methodologies to more naturalistic and 'artistic/designerly' forms of inquiry" (Gray and Malins 2004).

The concept of the participant observer was first developed by anthropologists and was later adopted by Design Studies, where participant observation is considered to be an uncommon, yet valid, research method (Blessing and Chakrabarti 2009: 247). There are different levels of participation. HCI researchers, for example, sometimes work as advisors or collaborators with users.

In Art and Design, the participation is commonly "complete participation", in that the designer is the researcher. "Practice-based" methods of enquiry emerged in the field of creative arts and design in the United Kingdom in the late 1970s (Gray and Malins 2004). This method has the artist or designer using practice as a vehicle for inquiry, generating knowledge through reflecting on their own practice. The researcher acts as what Donald Schön describes as a "reflective practitioner", developing thoughts about practice through engaging in practice.

These "research through design" methods have begun to appear in games research. Holopainen, Nummenmaa and Kuittinen, for example, used participant-observer design research methods to gather data on their design process for an experimental pervasive mobile phone game, in order to discover which core mechanics would work best in this game genre (Holopainen, Nummenmaa, and Kuittinen 2010).

Accordingly, the 2015 game studies textbook "Game Research Methods: An Overview" reflects this emerging trend, featuring a section of practice-based research in game studies, featuring a section entitled "Development for Research" (Lankoski, Bjork et al 2005). By this they do not mean games developed for the purpose of data-collection, or as test beds, as it is widely used for in HCI and Computer Science based game research (in fact it is rare these days to see games research using standard HCI and Computer Science methods that does *not* create a game or game prototype for the purposes of lab-based experiments or other form of validation). What is meant

here is the developing of games for observing aspects of the game development process and its outcomes.

The research projects discussed, however, have goals that can be even more narrowly defined than this, however: their research goal is to make some form of design contribution. As such, by "Development for Research" they mean, specifically, "practice-based" research. Although the terms "practice-based" and "practice-led" are sometimes used interchangeably in discussions of Art and Design research, they are not the same. While in practice-based (or "design-based") research the creative artefact(s) produced form the basis of the researcher's contribution to knowledge, practice-led research seeks to contribute new understanding about the practice itself (Candy 2006). My goal - a more interdisciplinary attempt to study tools in relation to the design process – makes my process "practice-led". This type of design-led research – and even specifically the evaluation of new tools and techniques – also has precedent as a method used in Art and Design. Malins in "Research procedures/methodology for artists & designers" and Bunnell in "Appropriate Research Methodologies" point to such research projects – ones in which practitioner-researchers have evaluated practice using, and integration of, new technology (Bunell 1998). Malins gives an example of a practitioner-researcher who evaluated kiln design through the creation of ceramics (Malins 1993); describing practice-based method as "naturalistic enquiry", Bunnell gives an example of a designer who evaluated computer-aided design using her own creative works as case studies.

The obvious disadvantage of the participant observer method is its inherent subjectivity. When the observer participates, they lose the advantage of distance and the objectivity that accompanies it. The participant-observer sacrifices distance in order to gain the benefits afforded by close observation.

Close observation is considered to be a key advantage of the participant-observer method. This is because external observation of design from a third party perspective (i.e. not as a first party, participant-observer) is difficult and poses its own set of risks to the integrity of the study. First, there is the problem of gathering data from designers. Lawson warns that conducting empirical work on the design process is notoriously difficult because the design process takes place inside our heads. Moreover, designers are not used to making this thought process explicit (Lawson 2006, 3rd revise:41).

Then there is the problem of the quality of that data. Designers, as compared with other practitioners, are notoriously unreliable test subjects when it comes to expressing their knowledge and methods. Schön warns:

> Designers know more than they can say, tend to give inaccurate descriptions of what they know, and can best (or only) gain access to their knowing-in-action [design knowledge] by putting themselves into the mode of doing (Schön 1992).

Describing elements of design process in a domain that still lacks formal concepts and vocabulary also contributes to the burden of difficulty and responsibility placed on the designer participating in evaluation research.

Another advantage conferred by the participant-observer approach is that it accommodates some of the practical issues of evaluating design tools. Any evaluation of design tools is dependent on the designer's knowledge, motivation and exploratory thinking. This means that, effectively, the designer is obliged to participate to at least some degree in the process as a researcher him- or herself. This is because, firstly, designing with unpolished, untested tools requires effort and a good deal of additional decision-making and commitment. Resnick et al. warn that users may have to "invest substantial effort" over long periods of time (Resnick, Myers, and Nakakoji 2005). Secondly, creative thinking is required: for example, working out what a tool could be useful for and how it could be integrated into a workflow for a given game project. As described above, tools that raise the design "ceiling", opening up new design possibilities that were not feasible without the use of the tool, could even require from the designer the creation of entirely new types of design tasks.

A participant-observer approach also helps avoid another problem third party researchers face: the misinterpretation and overlooking of important design events. Design discoveries occur rarely, making it virtually impossible for someone to be observing when a discovery occurs (B Shneiderman and Plaisant 2006).

Finally, any act of observation itself disrupts normal practice. This is minimised if the designer, as the observer, can regulate and control the disruption themselves.

The closeness and blurred boundaries of the participant observer method, therefore, can be considered its strength. The participant can be an informed observer, armed not just with

experience but with the analytical tools and vocabulary with which to identify, extract and frame elements of their design experiences relevant to the research.

5.6.3. A comparative study

Calling comparison "the main intellectual tool of analysis", Gray and Malins remark that it helps the researcher to form categories, establish boundaries, find inconsistencies, discover patterns and connections, and paint the larger picture beyond the specific detail (Gray and Malins 2004).

Where game design tools have received evaluation, it has been primarily with the goal of validating a research contribution. No comparative studies have been published; in fact, in the presentation of game design tool work it is rare to find even any mention of other game design tool research.

My research goals, by contrast, required a broad and comparative approach as opposed to a piecemeal study of each tool in isolation. To this end I evaluated several design support tools and models that vary in approach. I compare not just my experience with the various tools, but also my work without the tools during the study and prior to it within my professional practice.

5.6.4. Multiple case studies

My practical work has comprised the use of tools as applied to several game design case studies that range across a variety of game genres. This is largely motivated by the fact that game design problems vary across game genres and types. Furthermore, some of the tools under study are anticipated by their creators to be useful as applied to particular design tasks, game mechanics or game genres. My aim was to maximise the breadth and diversity of design contexts in order to maximise the potential for exploration and comparison.

5.6.5. In the field

Hewitt advises that creativity support tools be evaluated in the field, with users using them to do their real work, in real time, over an extended period of time (Hewitt 2005). This recognises the "situated nature" of most work activity, with participants carrying out their own work instead of a set of tasks supplied by researchers (B Shneiderman and Plaisant 2006).

Lawson contrasts studying design in experimental conditions with "normal" conditions. Asking designers to design under experimental conditions allows researchers to observe the process

closely and gain accurate results – but the design process is unnatural. Meanwhile studying design in normal conditions is also flawed because it sacrifices accuracy. Researchers are reliant on the memory and the descriptions of designers, who are liable to unwittingly misreport their process (Lawson 2006, 3rd revise:45). This problem is somewhat mitigated using a participant-observer method, as described above.

5.6.5.1. Balancing research needs with design needs

It is important that both research method and design method work together in a complementary, harmonious way. Good science means not allowing the act of data collection to affect the data in such a way that it is rendered useless. Ideally, observation should not impact the process being observed.

In the case of a study of design where tools are the focus, the needs of the experiment are in danger of coming into conflict with the needs of the design itself – i.e. a scenario where the designer tailors the designed artefact to the strengths and weakness of the tools, and is effectively "used" by their tools. Such an imbalance between designer and tool is considered bad practice in design. While a designer is always forced to constrain and adapt their design based on factors external to that design (time, resources, client requirements, technology), it is commonly regarded as bad practice to allow their design to be led by the needs, bias, strengths and weaknesses of their tools. Doing so often leads to bad design outcomes, such as design clichés, and work that bears a recognisable "watermark" of the tool that was used[18]. A good designer is expected to select and adapt their tools based on the needs of their design, not the other way around.

In a research context, therefore, avoiding the subordination of one's own design vision to the vision of the designer of the tool is all the more important when the aim is to simulate real-world design practice. Shneiderman and Plaisant advise that the evaluation of tools should not be forced, and that users should be encouraged to use the best possible tool for the (design) task, in order to "avoid a situation where users try to please the researcher by using the new tool while another classic one would have been more appropriate" (B Shneiderman and Plaisant 2006). They

---

[18] In the game industry, this phenomenon (which used to be commonly seen in game art portfolios of recent graduates), was known as "the rotating cube above a shiny black and white tiled floor" (something that is produced by a 3D modelling tool that does what the tool is best at: i.e. maximally impressive with the least possible effort or skill to produce). There are also running jokes like this in other creative industries, especially ones that use computer-aided design (electronic music being another good example).

further observe that even specifying tasks is at odds with the goals of supporting innovation or discovery (B Shneiderman and Plaisant 2006).

With this in mind, I made a conscious effort designing from the point of view of the needs of game design, rather than the point of view of the specificities of what the tools claim to or can offer a game designer. In general, the researcher (me) has been careful not to interfere too much with the work of the designer (also me). Only then can the researcher part of me hope to allay suspicions that the designer part of me is not merely telling her what she wants to hear

However, remembering Lawson's warning that any study of design thinking is necessarily flawed in some way, we may have to accept that conditions for the observation of design thinking can never be perfect, and the data can never be wholly free from influence from the research experiment itself. Even the designer's assessment of what is the best tool for the task, however, is not free from contextual influence. The choice could be influenced by the psychological effect of "familiarity bias" – i.e. a bias against the unfamiliar tool. One cannot control for the "newness" of new tools, when comparing a new tool with an older one. Added to this is the likelihood that, as compared with new tools the designer is less proficient using, familiar tools and methods are, in context, almost always going to be the best possible tools for that task for that designer, in that moment. So one could argue that the designer be prepared to sacrifice their short term design needs in order to invest in using the newer, temporarily inferior tool, where they feel they can anticipate some future design payoff. These and similar decisions and interpretations rely heavily on the design judgement of the designer, the designer needing to be self-reflective enough to be conscious of the biases and feelings that drive their tool usage decisions. In this way, the designer is having to "reflect-on-action" not just as a designer but to some extent as a researcher as well; the research is reliant on the judgement and prerogative of the designer. Observation and design, in this sense, overlap and become hard to draw a clean distinction between. In this respect, the benefits of the participant observer method become clear.

5.6.6. Longitudinal case studies

I used longitudinal case studies; that is, I ran my game design projects over months and years in order to study the tools. This gave me the opportunity to study the tools at various stages of design.

Longitudinal case studies are used in creativity support tool research as an alternative to controlled studies:

> It is still an open question how to measure the extent to which a tool fosters creative thinking. While the rigor of controlled studies makes them the traditional method of scientific research, longitudinal studies with active users for weeks or months seem a valid method to gain deep insights about what is helpful (and why) to creative individuals ((Seo 2005) cited by (Resnick, Myers, and Nakakoji 2005))

With the evaluation of design tools, the designer is not merely compelled to learn how to use an unfamiliar interface of perform an already familiar task or process (as is common in HCI research) but to learn and reframe their design thinking within the new formal constraints of a process that may be wholly unfamiliar to them. The designer must integrate the tool into an existing workflow, or to make any necessary modifications to that workflow. It may be necessary that the designer devise and modify design processes in order to prepare design materials for use in the tools. Specifically, I have found that using a tool requires preparatory design work, and that design outputs from the tool may have an indirect impact on the design activities that are undertaken after and alongside the actual use of the tool. Finally, the designer may find themselves having to devise entirely new design tasks by virtue of the tool opening up new design possibilities.

This starting point is even further back with game design tools as compared with tools for design disciplines that already use design support. By asking a game designer to use a game design tool, we are asking them for the first time in their practice to reframe and somewhat formalise their design thinking within the constraints of design process that includes the use of design support; in effect, to transform their practice from a vernacular, craft-based design practice to one of self-conscious design.

This all takes time. A longitudinal study addresses this, allowing time for the impact on design practice ("strategy changes" (Shneiderman and Plaisant 2006)) to develop and become apparent.

5.6.7. Data collection and observation

The data for this study were collected in the form a design diary, forming the basis for observation in the form of narration and discussion. This method for recording and analysing design practice has been found to be useful in in Art and Design research and by game designers themselves.

Data collection in the form of a design diary (as used by Holopainen, Numennmaa and Kuittinen, for example (Holopainen, Nummenmaa, and Kuittinen 2010)) is a popular data collection tool for

the study of the design process. Gray and Malins advise the designer, as participant observer, to analyse as they observe. They suggest that it is helpful to track this reflection in a journal alongside the data, as part of an iterative process where both data gathering and analysis serve to inform and drive each other (Gray and Malins 2004). Shneiderman and Plaisant also suggest the journaling of not just difficulties and successes but also insights (B Shneiderman and Plaisant 2006).

We should also note that game designers themselves have developed their own methods for collating and presenting observations of practice. While it is frequently observed by academic researchers that game design is a relatively young design discipline, video game design as a practice and community is larger and more mature than academic game research[19]. As such, their methods might have something useful to contribute to ours.

The most widely used method for the evaluation of game development and game development process is a method borrowed from the software industry: the "post mortem". A post mortem is a narrative in the form of a presentation or an article that describes the design and development process for a game, accompanied with analysis and evaluation. When conducted internally, its purpose it primarily for the developers to reflect upon their work and analyse their successes and mistakes in order to learn from them. It is also proved a popular method for sharing and disseminating design knowledge for the benefit of peers and newcomers – long before games education began to become formalized.

There are a few things we can note about game development post mortems. They are subjective. They are rich in context, emphasizing the specificity of the conditions under which development took place. They often include details such as personnel issues and changes, the past experience of the team, scheduling details, market conditions and requirements, marketing-related events, and so on and so forth. This could be seen as an acknowledgement of how the conditions and constraints of design are important because they are particular to each project, and their impact

---

[19] While Chris Crawford published *The Art of Computer Game Design* in 1984, for example, the first comparable academic contributions on the subject of game design only began to appear two decades later (Salen and Zimmerman's *Rules of Play* was published in late 2003); the Computer Game Developers Conference began in 1996, while the first Digital Games Research Association conference took place in 2001 – described as "year one of computer game studies" (Aarseth 2001).

on the development process creates differences that make them highly relevant factors. In order words, they are contextual details that readers feel they need in order to make sense of the data.

In this work my observations are, by necessity, mostly focused and on point. I have, however, born this in mind: that contextual information is sometimes relevant and important.

5.7. Conclusion

In this chapter I identified the need to contribute evaluation work that goes beyond simple research validation: we need third-party, comparative, practice-led evaluation of the work to date on game design tools in order to help move this research field forward. After establishing that there is no one correct method for the evaluation of design tools, but instead a variety of methods that can create "converging lines of evidence", I proposed a designerly form of enquiry based on longitudinal case studies.

6. OVERVIEW OF CASE STUDIES

6.1. Overview

Several games were designed and built for the purpose of evaluating how game design tools operate on realistic design problems. In later chapters I narrate and analyse my experiences with design support while creating these games. This chapter provides context for this: it gives brief descriptions of the final design for each project. I also summarise the design process for each, the elements of which are narrated briefly here but described and explained in more detail in later sections where I discuss in more detail my experience with the various tools.

6.2. Introduction

In the previous chapter I explained why I have chosen to prioritise the needs of design rather than adapting it to suit the particular features, strengths and weaknesses of the tools for the sake of observation. The advantage of this is that it better simulates real-world practice, and minimises the effects of observation on the process being observed.

The downside of this, however, is that being led by the design rather than the needs of tool evaluation is very inefficient. Indeed, staying true to this method in this project has meant carrying out a comparatively large amount of design (and production work in order to enable different stages of design) for which the tools under study proved not to be applicable at all. Where the tools were of no use, work was completed using conventional methods: using a combination of documentation, sketching, prototyping and (briefly) paper prototyping.

Though this consumed a great deal of time and did not generate any direct experience with the tools, it did contribute in a least two ways: 1) providing material for making comparisons between tools and the use of conventional methods after having become familiar with those tools (for example, I was able to compare my experience and relative success using a tool versus conventional methods on the same design task for a single game, and also between the use of a tool on one game versus the use of a conventional method on a different game); and 2) affording me the opportunity to see the interesting impact having been exposed to these tools has had on my design practice *even when I am not using tools.*

I did, however, make a conscious effort – up to a point – to maximise my exposure to tools in a general sense. First, I made a conscious decision to attempt, in most cases, games that feature design tasks that have been highlighted by the research community or designers (including the creators of the tools under study) as candidates for design support. One of the major hopes expressed by the game design tool advocates, moreover, is that tools could expand the creative horizons for game designers ("raise the ceiling" is the metaphor used in creativity support literature), enabling them to undertake more ambitious and complex design tasks that result in more sophisticated, interesting games. Second, I attempted to maximise the variety of design tasks I would be undertaking by aiming for variety in the game genres and styles I worked on.

At the risk of stating the obvious, however, this well-intentioned planning concerned matching game projects that had not yet been designed, with tools I had never used. Thus, even for projects for which the design was not particularly experimental in nature, predicting which tools would be applicable was only correct some of the time. In some cases, even a minor mismatch between tool and task blocked me and forced me to almost immediately abandon using a tool for a task. Following the needs of the design of the game wherever they lead meant that some features of the tools were left virtually unexplored (or at least not used practically). The un- and underexplored tool features could perhaps be usefully applied to the design needs of other games, or they may be features that I can imagine would have been useful applied to design tasks I have undertaken in the past; I am simply unable to say. Mismatches between my games' design needs and what the tools were offering could prove frustrating and unsatisfying at times, but while I was sometimes tempted to tweak an element of my design to be a better match, or go in a certain design direction to better exploit the seemingly exciting design directions offered by a tool, I did my best to stop myself from doing so. Fortunately, however, the design sometimes led me in directions that generated new tasks for which a previously non-applicable tool became useful.

In addition to the non-tool-supported design work on the five projects described below, I also worked on game ideas that did not bear fruit – in the sense that they did not prove "fun" and successfully pass much beyond the ideation and prototyping phase of design.

Early on in this work I spent a good deal of time on a language-based real-time puzzle-comedy game about speech-making. As the central idea was comedy-related, the "fun" of this game relied heavily on content, and so prototyping the gameplay was essential. My hope was that after coming up with core gameplay that worked, I could then employ design tools to help balance the

game. Unfortunately, however, even after taking the design in a number of different directions and building several prototypes, I failed to find a core mechanic that was enjoyable enough to build a game around. I now wonder if this was because I, like many others, was still under the influence of the prevailing "fail early"/"make a toy, then make game out of it"/game jam-style faith in starting prototyping before I had fully thought through my design.

Another dead end was a project motivated by the desire to create a persistent, economy-based two-player multiplayer game that would be a good candidate for game system modelling. Persistent games have long-term emergent properties that are difficult to model without computational support. As for the project described above, I made the mistake of prototyping too ambitiously, and after a great deal of time spent creating a server-side running online multiplayer game prototype I found the moment-to-moment gameplay to be unsatisfying and was forced to rethink the concept, leading to the initial idea for *South Sea Trouble*, described below.

Other abandoned work includes design directions for five projects that I pursued for a time until I decided for either design or production reasons to halt work on them – the "RTS lite" and multiplayer modes for the game *South Sea Trouble*, for example. Some of these directions involved the use of tools, and this work will be mentioned where useful.

It should be noted that, with the exception of *The Casimir Effect* (for which I collaborated with artist Dimitri Lecoussis, who also co-designed the game's narrative) the production assets for the case studies used low quality "placeholder" art. Though not ideal, visual production quality was not the priority or focus of this project.

6.3. The Casimir Effect
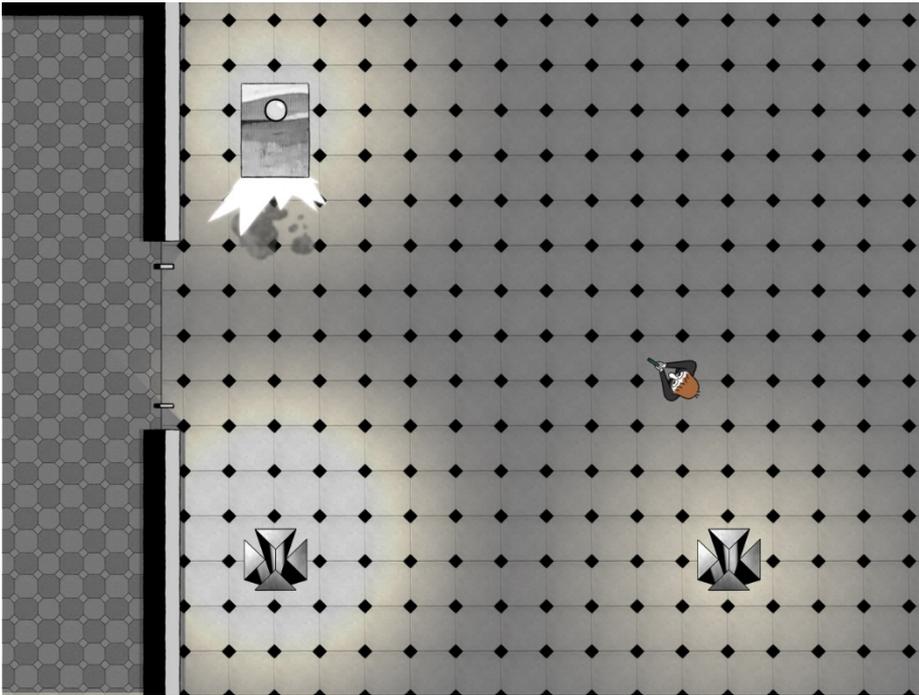
Figure n: A screenshot from The Casimir Effect

6.3.1. Design overview

The Casimir Effect (shown in Figure n) is a top-down shooter with an original combat mechanic and puzzle elements. All combat is ranged, with a single weapon and projectile type. While conventional shooter combat is possible, the player is afforded the possibility of enhanced, indirect combat via the use of "materials" – cubes positioned in the environment that amplify the projectile's damage, alter its trajectory, and allows the player to store up and time multidirectional, simultaneous attacks.

The game's level structure comprises a sequence of six major sections that are divided into 10-15 small levels linked in a branching structure. Progression is based on completion of these levels and sections as well as resource harvesting, which allows the player to replay and unlock branches (using a resource-based lock-and-key mechanism) towards previously inaccessible levels in sections they have already played.

The narrative is adaptive, evolving as a function of the unlock state of level branches.

107

6.3.2. Design tasks and challenges

The Casimir Effect, as an action game and a game of progression, requires the creation of a significant amount of game content in which the player is able to acquire and master skills. The design of this content – known as "level design" by practitioners and "progression design" by some researchers – is a key design task. A large number of combat scenarios and puzzles must both be conceived and arranged within the level structure.

Like other games of this type, *The Casimir Effect* does not typically feature complex emergent dynamics. The branching structure, however, and especially the replay and unlocking scheme described above, tied as it is to a resource-harvesting-based game economy, renders the progression design a complex task.

Designing content for the adaptive narrative is also a challenge that extends beyond the usual narrative design scope of an action game.

6.3.3. Overview of the design process

First, I ideated the basic combat mechanic of using the above-described "materials" to suspend, reorient and amplify projectiles. To aid this I drew literal (i.e. simplified versions of what would be displayed on-screen) diagrams describing the moment-to-moment gameplay of this mechanic. I then prototyped and playtested it, and also variations of it in order to explore how much gameplay content could be extracted from this single idea. Once satisfied with this, I prototyped a sample level, which I recruited friends to playtest. Upon receiving positive feedback I then decided to go ahead with designing a game based around this combat mechanic.

Prototyping continued alongside my design work with tools. The prototype allowed me to refine the pacing, action and interaction aspects of the core combat mechanic.

I used a tablet and a stylus (i.e. pen and paper) to make literal sketches of puzzles and devise new gameplay elements.

Meanwhile, I attempted to devise a lock-and-key level structure using Dormans' Mission/Space conceptual framework and his associated tool *Ludoscope*'s procedural content generation features. As the design changed and the unlocking scheme became linked to a resource-harvesting-based economy, however, I switched to trying to model this economy in *Machinations*,

before finally realising that the task required a means of integrating these resources with my level structure in some way. Sometime later, Butler's work on "progression planning" (Butler et al. 2013) was published and this ultimately became an approach I adopted for my larger scale progression design tasks.

I later began to use *Ludoscope* and its iterative model transformation approach at a lower level of granularity: to generate level content. The effect of this and my work creating a skill chain (Cook 2007) imposed a formal framework on my pen-and-paper sketching design activity in order to generate material for use with these techniques: I created a large collection of simple and "compound" (literal) puzzle patterns. These I later linked back into my larger scale progression planning and used as a basis for creating content generation rules for *Ludoscope*.

To design the game's narrative I initially used *Microsoft Visio* (non-domain specific flow-charting software) to diagram and visualise its adaptive, multi-state structure, with a view to using this layout and visual language to create and edit the story itself. I then briefly attempted to use *Articy:Draft* for this task, but quickly failed to achieve a better result. Eventually (and ironically) it was a much simpler representation in *Microsoft Excel* that proved the most effective for developing and communicating the complexities of the story.

6.4. The Particle Who Knew Too Much

6.4.1. Design overview

The Particle Who Knew Too Much is an adventure game designed within a hybrid text/graphical adventure game system I devised for a previous game, *Alone in the Park* (2011). In this system, conventional point-and-click gameplay is replaced with a combined conversation and inventory system (similar to the "memory" system in *Touch Detective*) while the game's action is represented by text that outputs in real-time in response to player action within a 2D map-based game world.

As I had previously devised the main gameplay system, most of the work in designing *The Particle Who Knew Too Much* involved creating a new narrative and new quests within the constraints of this form, with its set of existing game mechanics. I did make one addition to the system, however: the introduction of an in-game currency, to help manage game progression and allow a richer palette for the creation of multi-solution quests - gameplay more often seen in open-world action

RPGs than adventure games. In *Alone in the Park*, quests had one solution[20] and were, for the most part, self-contained. Yet even simple, slightly non-linear quests with few interdependencies gave rise to a wide scope of (sometimes unanticipated) game states, difficult to understand without the use of diagrams, and requiring a not insignificant amount of design management of the game's possibility space. For this project I wished to see whether design support could help me manage an even wider possibility space, and facilitate the design management of an even greater possibility space arising from multi-solution, interconnected quests.

6.4.2. Design tasks and challenges

Adventure games are games of progression, but this progression is typically somewhat non-linear. Much like "open world" action games or RPGs, an adventure game has a partially open-world format where the player can move at will between several locations, working on several quests in parallel, the tasks for which do not necessarily need to be completed in a set order. Therefore, while the gameplay is simple and does not give rise to complex emergent dynamics, it is the game state in terms of game progression that is hard to predict. In terms of progression, this means a larger (hence a more complex) possibility space for the designer to manage as compared with linear game genres, such as a puzzle game with a sequence of levels to complete, or an action-adventure game like *Tomb Raider*.

The addition of a currency system offers the designer the ability to abstract and universalise logical dependencies so as to offer flexibility and manageability for gating game content (see Chapter 15 for a definition of gameplay gating) for the sake of game progression. A currency system requires modelling, however.

A multi-solution, interconnected quest game structure that uses currency gates is a complex logical network that poses a challenge to document and think through.

Management of game progression and game state does not resolve narrative interdependencies in the game's fiction. The player's "knowledge" state must be logical also, requiring careful planning of the narrative design.

---

[20] This is typical of nearly all adventure games; in fact, being "blocked" by puzzle "unfair" bottlenecks is a common complaint among adventure gamers. Arguably, it is even partially to blame for the demise of the adventure game as a genre in the late 1990s.

6.4.3. Design process

As the core gameplay had already been designed in the previous game, the design of this game began with the development of narrative ideas.

The tools used to document these were *Evernote* (Evernote Corporation 2008) (virtual notebook software), *Articy:Draft* and *Microsoft Visio*.

I also used *Articy:Draft* for content production tasks, combining my "thinking" workbench design activities with content production (the reasons for which will be discussed in later sections). I was able to do this with a few modifications:

My previous game supported a data-driven production approach, in which I used a graphical XML editor to edit quest and narrative data in the form of XML files, which were then loaded and interpreted by the game executable. Therefore to create a new game it is primarily a case of creating a new set of data.

I used *Articy:Draft* to create my game's data types, create data with them and export narrative and gameplay data, and wrote an importer to use these data in my game. Additionally, I extracted some of the remaining hardcoded game logic from my code and translated it into additional data types so that essentially all narrative/gameplay content for my game could be produced using this tool.

In the way described above, I used *Articy:Draft* as a documentation, visualisation and game asset production and editing tool. In terms of the relationships between the quest and narrative data, however, I applied computation support in the form of my progression design tool (which is described and discussed in Chapter 12) in order to deal with the progression design challenges described above.

6.5. *South Sea Trouble*
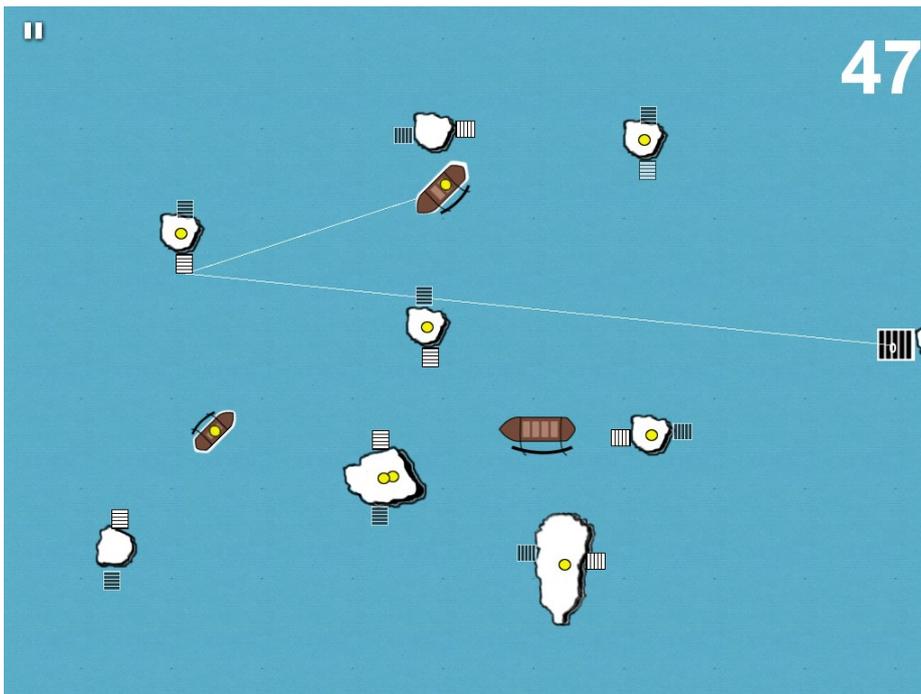
Figure o: A screenshot from South Sea Trouble

6.5.1. Design overview

South Sea Trouble (shown in Figure o) is a casual, twitch-based, strategy-oriented puzzle game in which the player coordinates a small fleet of boats of varying speeds and sizes to pick up and deliver passengers from islands to a destination within a time limit. Though the game is real-time, a mechanic I am calling "command-queuing" allows play that creates an experience very similar to plan-and-execute gameplay.

The player progressively unlocks a loosely branching sequence of levels, where a level is a screen-sized map containing a unique arrangement of islands, reefs and boats. The level sequence is loosely branching, ensuring that not all levels must be completed for the player to progress. Each level is introduced by a brief narrative that offers a fictional premise for undertaking the level. These stories tie into an overarching game narrative.

As the player's skill improves, the player is encouraged to go back and replay levels with the aim of completing them more quickly to achieve higher scores. Replay is also encouraged by a system of collectibles: time saved on collecting passengers can be used to pick up collectible items. These items form sets and are displayed on separate pages in the game's front end.

6.5.2. Design tasks and challenges

Level design – the contents and spatial layout of levels – is a large task for this game.

Progression design was required to create a path for skill acquisition, increasing challenge and complexity. As is typical of a level-based puzzle game, the game's full repertoire of actions and puzzle-solving techniques must be progressively introduced to the player at an appropriate pace.

Narrative design tasks comprise the design of the main narrative, the level narratives (some of which may not be seen by the player as the level may be optional), and player and non-player character management.

Though no longer part of the current design, the design of a game economy was required for the resource harvesting and building mechanics ("RTS lite") of an additional game mode.

Also no longer in the design was a multiplayer mode that required further level design work with an additional challenge: balancing the spatial positioning of island and passenger placements on the map for two players.

As a twitch-based puzzler with extremely repetitive interactions, the design and polishing of the game's interface and control scheme is core to the design work of a game of this genre.

Balancing elements of the core gameplay – balancing speeds, boat capacities, timing, and entity sizes – was required.

6.5.3. Design process

From an initial idea I prototyped and modified several iterations of the core gameplay. Complementing this process I devised and calculated modifications and improvements relating to spatial and timing factors (boat capacity, speed, distances) using pen-and-paper sketching.

Alongside the main game mode I prototyped additional modes based on the same core gameplay but offered different objectives and additional mechanics. For some of these I used *Machinations* to think and "play" through the system mechanics of these ideas before prototyping them. I found that while often seeming successful in the tool, they were unsatisfying to play when prototyped. Even where prototypes showed promise, I decided that they strayed too much from the "plan-and-execute" pace and gameplay style of the main game mode (which had resonated well with

playtesters), and that there were many possibilities for more subtle variations on this game mode. One of the modes prototyped, however, I felt was too successful to abandon and this I took out of *South Sea Trouble* to work on as a game in its own right (see *Ultraworm* below).

The abandoned modes included a multiplayer mode. This mode was partially motivated by the idea of being able to make use of *Sentient Sketchbook*. After having created a turn-based paper prototype of this mode, I created a local multiplayer prototype of this mode. I then realised that I would require online multiplayer support (a feature that my game engine did not at that time offer) to feasibly have enough opportunity to playtest and iterate on the prototype. I did build levels based on designs I made using *Sentient Sketchbook*, but as the multiplayer gameplay itself needed design work it was hard to know how successful they were. Not being able to customise *Sentient Sketchbook* was also a blocker. Finally, I came to realise that perhaps a more promising direction for multiplayer would be a collaborative mode – a style of multiplayer gameplay that *Sentient Sketchbook* was not designed to support. The substantial production work required to create a multiplayer mode seemed to offer little research pay-off and so I decided to suspend work on multiplayer mode. Unfortunately, this meant that beyond what I have recounted here, my limited experience with the tool has not given rise to observations to include in the observation and analysis sections of this thesis. However, even this minimal experience has no doubt contributed to my practical understanding of mixed-initiative design process.

Once the game design had become near final I began to use my progression tool to map out levels and order the introduction of skills, techniques, game mode variations and entity types.

Meanwhile, for planning the game narrative and documenting and managing my narrative ideas I used *Articy:Draft*.

6.6. *Loot the Room*

Figure p: Screenshot from Loot the Room using placeholder art

6.6.1. Design overview

Loot the Room (shown in Figure p) is a casual puzzle-strategy game. Aesthetically, the game could be described as a casual, sped-up dungeon crawler. The player directs a team of characters (Viking looters) through a maze-like dungeon-style space (a monastery) to loot rooms for treasure. Rooms are different sizes, and their sizes correspond to the number of looters that are required to be in the room at the same time in order for the room to be looted (i.e. 1-looter rooms, 2-looter room, etc.) A looter will always attempt to loot an unlooted room that they pass on their way through the maze. The player repeatedly sends characters through the dungeon on different paths, attempting to coordinate the characters' looting efforts, while accessing pickups and circumventing obstacles. The player controls the game very simply by tracing a given route for a given character through the maze.

Like *South Sea Trouble*, the game comprises a loosely branching sequence of levels, a level being a dungeon. The player must loot as much gold currency as they can from a dungeon within a time limit. To advance, they must achieve a set amount of gold for that level.

Between levels the player is given the opportunity to improve their team's looting capabilities by spending their currency: to hire new looters, upgrade characters and buy special items. There are

115

three classes (with upgrades this creates 10 subclasses) of looter characters, which vary in their attributes of speed, loot-carrying capacity, strength and magic (similar to RPG character classes). Before each level starts, the player has the opportunity to study the level layout and contents, and select the right balance of team members to deploy. In this way, this meta-game economy adds a strategic dimension to the moment-to-moment spatial puzzling.

6.6.2. Design tasks and challenges

Level design - designing the layout and content of each level - is a major design task for this game.

As for *South Sea Trouble* the progression design for this game required creating a path for skill acquisition, increasing challenge and complexity. In addition to the level structure, however, progression also occurs within the evolution of the player state: the player's resource allocation choices vis-a-vis spending currency to build their team and advance through character upgrades. A variation in player state impacts the difficulty of a level – some choices could even have the effect of blocking progress, for example. This renders the task of designing the game's progression a multidimensional challenge.

The game economy itself – rewards, prices, currency distribution between levels, etc. – must also be designed.

6.6.3. Design process

Initially I conceived of and refined the core looter coordinating idea on paper through sketching out the moment-to-moment gameplay. I then began prototyping, iterating through various maze sizes, speeds, number of characters, gameplay tweaks and so forth until I settled upon a format that was solid enough to lock down. I had also begun to sketch design patterns/level design building blocks that could offer the kinds of puzzles and challenges to be solved by the use of these coordination and repeated looting gameplay mechanics.

I created and iteratively edited some level designs using *Ludoscope*'s procedural content generation system.

Machinations was used to model the team upgrading and currency based aspects of the progression, and generate a variety of possible scenarios for player state evolution. This data was

116

then input into my progression design tool as player state estimates to map against level progression.

6.7. Ultraworm

Figure q: Screenshot of *Ultraworm*

6.7.1. Design overview

Ultraworm (shown in Figure q) is a casual real-time puzzle game where the player manages a fleet of vehicles to collect and deliver different coloured resources to their similarly coloured destinations. Resources spawn randomly at unoccupied locations at a slowly increasing rate. Clearing a spawn point of its resource and delivering it to its destination earns the player a point, the objective being to score as many points as possible before the game ends. This occurs when there are no occupied spawn points left for a new resource to spawn. Using the lens of *Patterns in Game Design* (Björk and Holopainen 2005), *Ultraworm* is an "unwinnable game", where difficulty within a single level increases continuously until such time as the player makes a mistake and "loses", at which time the level ends.

This core gameplay and scoring is enlivened with the addition of disruptive mechanics that periodically inject high-risk, high-reward limited-time objectives. These mainly appear in the form of pickups (Match-3 style bombs, speed-boosts, and coins) as well as bonus conditions (e.g. for a few seconds blue resource deliveries receive a bonus).

Ultraworm contains mechanics that consider the player's experience with the game over time – known as the "games-as-service" model, common to popular mobile/tablet games. One of the aims of games-as-service – especially for highly replayable games of this kind - is to encourage repeated play and daily play sessions. To achieve this, an in-game currency is usually used: progression is treated as an abstract resource which can be awarded external to in-game play.

Like a conventional console or PC game, *Ultraworm* consists of a sequence of levels, which introduce both these core and additional game mechanics progressively. This progression sequence (in the form of new game levels), however, is unlocked almost wholly via the use of currency (this differs from the use of currency in the other games described above, in which in-game currency, where it is featured, is only a partial and indirect contributor towards unlocking progress). Currency is earned by the player in three ways: 1) by receiving currency rewards for attainment above a certain score when playing or replaying an unlocked level (up to a daily maximum reward); 2) by harvesting coins during play in the form of limited-time pickups; 3) by receiving a once-per-day coin bonus for simply launching the game.

Another typical design goal of contemporary mobile games is social play. As well as keeping track of personal high scores, each level of *Ultraworm* level is associated with a leaderboard.

6.7.2. Design tasks and challenges

One task is creating the progression sequence. This means balancing the introduction and the parameters of the core mechanics (star spawning frequency, ship speed, number of ships, capacity, resource spawn points) and the disruptive periodic mechanics (described above) for their influence on level difficulty, level completion time, scoring and currency accumulation.

Scoring is especially important in a game of this type. Achievement and skill needs to map to scoring outcomes in a logical and fair, yet dynamic and exciting way. This is achieved by finding the right balance between pure skill and grind versus random and disruptive elements.

The currency reward system and how much each component of it delivers over days and weeks needs to be modelled in relation to an optimal number of replays and daily returns, as well as how smoothly and at what average rate (and range of variation thereof) it allows the player to progress.

6.7.3. Design process

Initially, *Ultraworm* was prototyped as a potential new mode for *South Sea Trouble*. From playtesting I could see though that it was a very different kind of game experience and could be taken in its own direction to become a standalone twitch-based puzzle game. I continued to experiment with and refine interaction and pacing within the context of the prototype.

The progression design depended heavily on balancing difficulty between levels, while making sure that the shape of the dynamics over time within each level is still interesting. For this I modelled the mechanics as accurately I could in *Machinations* in order to figure out the balancing data for each level. I then recorded the data for each level in my progression design tool. As *Ultraworm* contains a small number of levels in a linear sequence I used progression planning mostly as just an organisational aid, as far as this difficulty balancing data was concerned. I used the "exploration mode" of my progression design tool to rehearse level replays in relation to: accumulating currency rewards and gameplay gating, scoring, and overall play time.

6.8. Conclusion

The five case studies included in this project showed predominantly mission and level design needs, as well as some system design and some narrative design requirements. The nature of these needs differed according to the case studies' various game genres.

My methodological decision to be led by the somewhat unpredictable needs of the design rather than let the needs of tool evaluation steer the design process rendered tool selection itself a process of discovery, organically revealing the kinds of tasks that a given tool may be useful for.

Some design tasks were more common than others, with the result being that some tools were used more than others. For example, I found myself finding a use for Progression Planning for nearly all the games despite the variety of genre; meanwhile, owing to the fact that The Sentient Sketchbook targets a very specific task for a particular feature in a particular genre of game, I was unable, in the end, to explore it in any depth. As most game genres have some form of narrative elements, I was able to *Articy:Draft* for most of the case studies. My case studies did not require as much system design as that of some of the more system-oriented game genres – e.g. simulation, strategy and many board game genres. This meant that, even though I had some system design

tasks for which I used *Machinations*, I was unable to explore it in as much depth as if I had concentrated on those genres.

7.   OBSERVATION & ANALYSIS PART 1: THE FORM OF REPRESENTATION

7.1.   Overview

This is the first of three chapters in which I analyse and discuss some of my experiences working with game design tools. The discussion is framed around themes and evaluation criteria identified by the design tool and creativity support tool literature, and draws upon my observations of my experiences using game design tools. In these chapters I have found it useful to occasionally draw comparisons with design tools and practice in electronic music and audio, as this is the other design domain that I have most experience practising in.

At the end of Chapter 3 I argued that the key weakness in game design practice is our inability to represent the design situation. Accordingly, the question of how useful a representation for design thinking game design tools provide becomes central to my evaluation.

One of the key functions of a design tool is to allow the designer to create an external representation of the design situation. In this chapter I look at the form that representation takes.

In this chapter I offer five observations selected from my experiences. These describe: visual representations of a game design situation using *Articy:Draft*; shifting focus within the representation from mechanics to dynamics in *Machinations*; working on several layers concurrently thanks to PCG; nesting to facilitate working a high level in *Articy:Draft*; filling the gaps in tool representation with documentation, sketches and prototypes.

7.2.   Visual cognition of game design problems

As discussed in Chapter 2, in game design today, natural language, along with prototyping, is the primary vehicle used for representing the design situation. Many advocates of game design tools have pointed to the limits of natural language for describing the system mechanics of gameplay, and have proposed that we find a means of representing the design situation visually.

This sentiment seems to be confirmed by the apparent preferences of practitioners. In his interviews with game designers, Hagen recorded game designers' dissatisfaction with text-based

documentation and an increasing preference to express designs in visual form, including diagrams, posters, reference images, mood boards, animations and concept art (Hagen 2011). This is, according to Hagen, partly because of the constraints of the rapid iteration cycles of modern game development, but also because of an explicitly stated preference for visual over textual communication of design ideas. These visual means allow a design to be more effectively communicated to the development team than with a written document, but we could infer also that a kind of conversation with these materials must also be in operation.

Based on discussions with four game designers on the subject of what designers might want from the tool that researchers were building, Nelson and Mateas describe how two of the designers interviewed expressed interest in a tool for automated reasoning, while the other two were mostly interested in the idea of representational functionality a tool could provide: in the case of Nelson's tool, the ability to visually represent game mechanics – in other words, "design drawing" (as discussed in Chapter 3). One of these designers articulated his desire for a graphical design language featuring sets of built-in vocabulary for common design domains, and both were interested in their tool's formal, abstract model of game mechanics as a useful representation for reflecting on their designs (Nelson and Mateas 2009).

Expressing game design visually, however, is a greater – or at least a different – challenge as compared with doing the same for a visual design discipline such as architecture. Representing a game design visually requires us to create a graphical representation of non-visual problems. While this may pose a challenge, if we are to believe the Design Studies literature, this would not only be feasible but would be of benefit to game designers. Many studies have emphasized the importance of drawing and visual thinking in design (Do and Gross 1996), and design researchers have emphasised the benefits of visual cognition in design even for fields where design outputs are not visual. According to Goldschmidt, "imagery" as part of visual cognition can be amplified by the activity of sketching. In other words, creating externalized representations that take graphical form can enhance visual thinking as a mechanism useful for design (Goldschmidt 1994).

7.2.1. Observation 1: Visual representations of a game design situation

Certainly in the case of many elements of games – dialogue trees, branching narratives, non-sequential game mission structures, for example – graphical, spatial representations have advantages. Often the information about the relationship between one idea and another is as

important as the idea itself. Space helps visualise a whole layer of relationships among the elements within the design situation that would otherwise be hidden within the linear confines of a text document. As noted in Chapter 2, game designers already create visual materials (flowcharts, posters, diagrams) to take advantage of this. Game narrative and quest structure is often externalised by game designers in the form of flow charts (using *Microsoft Visio*, commonly).

Articy:Draft, in offering visualisation and organisation for primarily narrative and mission design materials, has used this as a starting point, using flow diagramming as primary interface of their tool. I found its visualisation to be successful: I could quickly visually "parse" my narrative flow.

For example, I was able to successfully map out my branching game narrative for my adventure game *The Particle Who Knew Too Much* as well as for the more-or-less linear *South Sea Trouble* (see Figure r).

Articy:Draft has created a domain-specific environment. Visual aids are offered, in the form of customisable colours and icons. I found these aids useful for visually parsing my structures, and for maintaining an informal but consistently applied visual "language" for my entity types. My entity types frequently evolved or changed over the course of the design, so it was useful to be able to use the loose framing afforded by very simple and easy to modify visual elements like colour. A designer could do this in a generic tool (e.g. *Microsoft Visio*) but *Articy:Draft*, as a domain-specific tool, has relieved some of the design thinking involved in doing so by having devised a set of visualisation solutions.
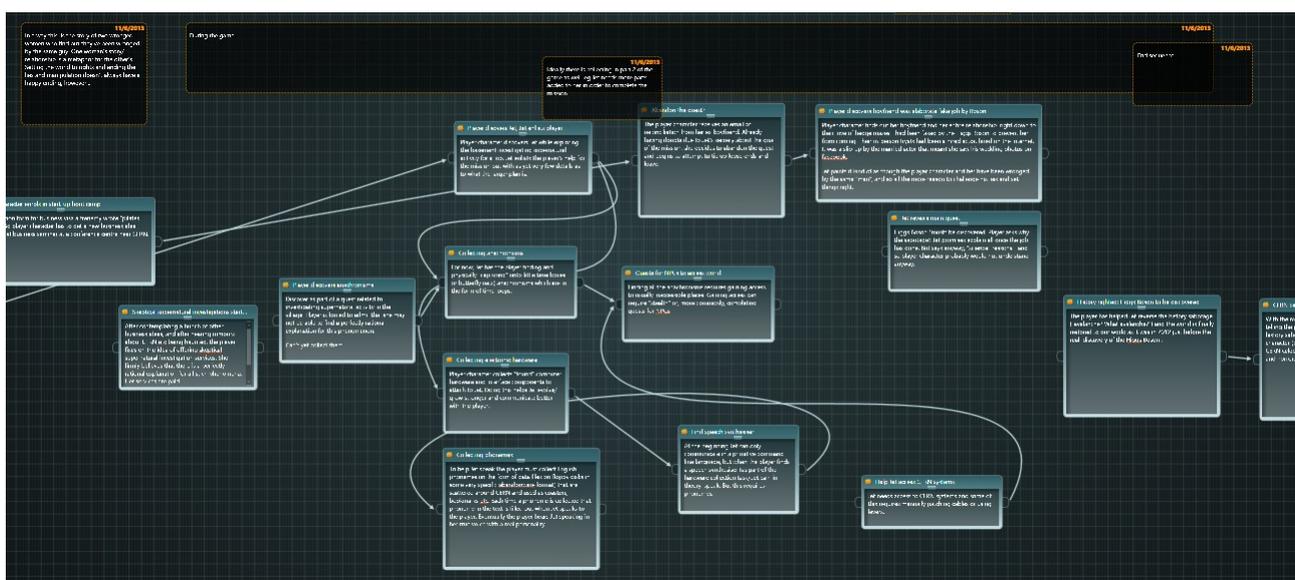

Figure r: Fragment from the narrative plan for *The Particle Who Knew Too Much*, using *Articy:Draft*

7.3. Language of abstraction

When writing a description of a game and building a prototype for that game, the designer makes choices: which elements to describe and to which level of detail, which aspects to include in the prototype. There are no difficult decisions to be made about the representation of these elements, however: a game experience, though it can be simple and cut down, is not abstracted – its point is to emulate a real, playable experience. One could say it is a "direct" representation of the playable experience; it is not an experience mediated by an abstract language that the user has to interpret.

A language or tool for the graphical representation of non-visual ideas, however, requires additional choices to be made. First, these are made by the tool designer: choices about the form and level or levels of abstraction, which and how much content to be included in the language. Then there are choices about the representation that are made by the designer using the tool. Unlike prototyping, a tool adds an extra layer of thinking and decision-making to the act of representing for the game designer.

For example, *Machinations* treats game systems as the flow and processing of resources. The designer must decide what elements of their design could be translated into resources. For some elements, such as money, the relationship between the element and its abstraction as a resource is obvious and direct. But representing more abstract concepts, such as game state or player progression, relies on the discretion of the game designer to devise the most useful representation. These representations may be less direct and more abstract, in terms of the relationship between the elements and their form of representation within the model, than that of a game element that more closely resembles a resource in the game (e.g. money, fuel, ammo). Figure s, for example, shows a diagram with relatively indirect and abstracted relationships between their representations in the model and in the actual game. The diagram models the time spent away from work on the main quest to complete side quests versus the speed gain on completing the main quest due to increased player skill, items, and so forth. Game elements such as quest completion progress and difficulty are represented as resources.

The game design tools in my purview abstract the same or similar content in different ways. They represent different, but often overlapping, subsets of the design situation. For example, *Refraction*'s progression planning tool focuses on concepts and player familiarity and skill with concepts, while progression design with *Machinations* focuses on resources being available to unlock progression stages. In *Refraction*'s tool, and in Cook's skill chains, game elements are enumerated but not quantified, while *Machinations* favours representing game elements as quantities of a limited range of resource types.

In this way, we could say that *a design tool represents a filtered view of the design situation*. The language of abstraction is like a filter design, with some parts of the signal coming through clearly and strongly (a small degree of abstraction), their importance even amplified. Other parts of the signal are rendered with a larger degree of abstraction and are consequently less clear and direct, or even filtered out entirely.
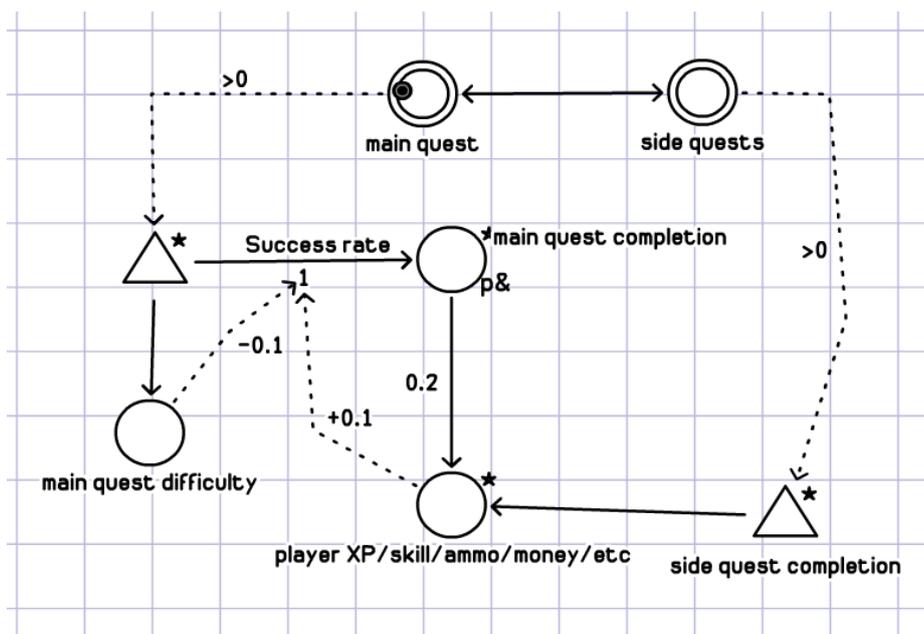
Figure s: *Machinations* diagram modelling relatively high level, abstract elements as resources

This matters for the goal of enhancing the designer's understanding of design problems using visual cognition. If the representation of a part of the design situation is too abstract, the benefits of visualisation break down. I have found at times that, for whatever reason (language literacy, failures on my part in terms of finding the best means of representing within the language), there

is too much of an imaginative leap to be made and too much to hold in short-term memory and this has a significant effect, reducing the value of the representation.

One of the founding goals in game design tool development is Doug Church's idea of a "shared language" for game design, with all the benefits that could bring. A shared language does not come without a cost, however.

There is always a cost-benefit trade-off between generic and specific languages, as researchers working with Domain-Specific Languages will attest. The question we have to ask is whether, in an attempt to create a universally-useful shared language, a model created by such a language is no longer useful as a representation of the design situation.

First, because of the form of representation: being generic may necessitate a high degree of indirectness in the abstraction from the concrete. Second, because the level of abstraction: being generic may necessitate working at such a low level of abstraction that the designer is compelled to build large, elaborate diagrams every time they want to make a design move. (See, for example, the level of detail in the diagram shown further in this chapter in Figure t.)

Below I discuss this latter attribute of the form of representation: the level of abstraction.

7.4. Level of abstraction

A tool and its language invites the designer to express the design situation at a specific level (or range of levels) of abstraction[21]. Indeed, one of the benefits of representing a game design

---

[21] "Abstraction level" could be taken two ways. First, from the point of view of player experience: similar to the concept of "abstraction layer" in computing, the position of the element on a continuum from game system design (low level) to elements that have a more immediate and direct relationship with the player experience e.g. enemy spawn-point placements (high level). Second, from the point of view of design (similar to Lowgren and Stolterman's abstraction layers): the level of detail or granularity included in the representation of the design situation, from the

situation outside of the context of production is gaining control over the level of abstraction of the representation. Nummenmaa et al point out that one of the powerful attributes of game design tools is that they can allow us to be able to gain a "broader view" of the game (Nummenmaa, Kuittinen, and Holopainen 2009). This is something a game prototype – which is typically a "vertical slice" of gameplay – cannot show us. We might call the level of detail – high or low – at which a game design tool allows us to view a design situation a "horizontal slice". Even a complete game does not allow us to "zoom out" from the level of moment-to-moment gameplay to see the global shape of a game over time like an abstract representation of it potentially can.

This ability to see the shape of longer term dynamics is important for games that feature emergent dynamics and indeed, it is Dormans' hope that his tool will equip designers to tackle the difficult design challenges that games as complex systems present. Prototyping is not well adapted for evaluating the design of these kinds of games. The prototyping of such a game - where progression relies more on the emergent dynamics of the game system than the design of a sequence of game missions or levels - could not be a "vertical slice", because gameplay cannot be produced without implementing more or less the entire game system.

Using the metaphor I proposed earlier (the tool as a kind of filter, in that it offers only a filtered, partial view of the design space) we could say that in addition to the form of abstraction, another attribute of the "filter" of the design space is the level of abstraction that it represents.

One layer of abstraction that these tools cannot easily represent is the highest one: "game feel", for example i.e. the fun of interaction and sensory experience (though to a limited extent they can let us "play" their simulations; this is discussed in Chapter 9). *Machinations* it is more difficult to see the content and shape of the game at a high level of abstraction in the way that, for example, *Refraction*'s tool's progression design approach affords. If, depending on the nature of the game

---

finer details (low level) up to broader shapes of the game (high level). Here, because I am talking in very general terms in which the distinction is not important, the dimension of abstraction I am referring to simply varies according to context.

concept, the content within a design situation that holds the key to fun is obscured from view, the designer may fail to see it.

Thus a tool cannot necessarily provide support at the level of abstraction that suits a particular game. In particular, this poses a problem for the notion of game sketching. Sketching, as we may recall from Chapter 4, is a rapid process that facilitates exploration – ideally the designer should be able to explore the design space in search of the core of a good (fun) idea around which to design a game. If parts of the design space are excluded from this exploration, some potentially good ideas could be missed.

7.4.1. Navigating abstraction hierarchies

Lawson says that that while in theory it may seem logical that a designer proceeds from high-level outlines to specific details of a design, evidence from studying how designers work reveals the reality to be far messier and non-linear. In architecture, detailing is sometimes done at the beginning in order to inform the shape of the overall design (Lawson 2006, 3rd revise:39). The order in which a designer works on various levels of abstraction not only varies between designers but also from project to project:

> What might seem a fundamental early decision on one project may seem a matter of detail which could be left to the end on another (Ben Shneiderman et al. 2005).

This is born out in game design, where a game system can be built around a single mechanic (e.g. a game like *Portal*) that is the game's "hook" (Hagen 2011). Dormans echoes this, saying that in practice there are many different ways of designing games. Here he talks specifically about a choice of starting point for design.

> Some designers might start with a premise, design rules to go with it and then proceed to levels and detailed stories. Others might start with a map that constrains the design of game mechanics. Even within the process of designing a game, steps to create particular parts of the game might differ. A tutorial level requires that the game mechanics are clear and finished, but other level content might be dictated by a storyline rather than game options. (J Dormans 2012: 163)

These choices about which are to be considered fundamental design decisions made early on in the project are not solely down to the personal design style of the designer. Very often they relate to the nature of the game, and where the key element, or "fun", or "hook", is situated. As discussed in Chapter 2, this can vary greatly between games. The fun of the game experience

could be situated at any abstraction level or levels. It could be in, for example, the pleasure of simple "twitch" interactions – the fun of timing jumps in a platformer, or targeting in a shooter, or the simulated stroking of an animal, or it could be the fun of collecting and managing resources towards a longer term goal. The former kind of fun would be located at what Ben Cousins might call the level of basic atoms of moment-to-moment gameplay (Cousins 2004b); while the latter is fun that would need to be modelled at a high level of abstraction.

The fact that the location or locations of what the designer regards as the core fun – and thereby often the design starting point or starting points – in a game concept can vary creates a dilemma that we see in game prototyping. A game prototype could feature a rough, but mostly functional version of gameplay, featuring rough "placeholder" art, but if the game idea is very much tied to the "look and feel" of the game, this may not do the idea justice. For many games the interest of the player is held by the sensory experience; indeed, too much attention-drawing by the mechanics of the game could distract the player's attention from the finer, more aesthetic details. I have heard Costikyan's piece "I Have No Words But I Must Design" referenced by those wishing to warn their fellow game designers that they should not (at least not systematically) underestimate the aesthetics as they can change the player experience at a surprisingly fundamental level (Costikyan 2002). It is for this reason that prototypes are often accompanied with concept art in order to achieve a more complete representation of the design situation.

In this context, where game designers might see a set of dynamics or aesthetics as the key concept and starting point for their game design, it is no surprise that they might want to work backwards from these dynamics to the system mechanics.

> …designers were particularly interested in "backwards" reasoning from outcomes to mechanics changes, e.g. what the smallest or largest value for a particular quantity (health, sword strength, etc.) should be to still achieve a desired outcome (Nelson and Mateas 2009).

We can take from this that tools that allow the designer to approach their design problem from different angles and starting points, at their choice of abstraction level, etc. can help accommodate the specific needs of a given project, or the personal design style of the designer. This is enabled by affording the designer a degree of freedom to move from activity to activity, and abstraction level to abstraction level within the design space, at the designer's discretion.

We often do not enjoy this freedom in current practice, particularly within large scale productions. After a pre-production stage during which prototyping occurs, we are often unable to freely make these decisions about starting points for design. Navigating freely through the abstraction hierarchy from detail (low level design moves) to broad brush strokes (high level design moves) would be too costly. Design details are typically fleshed out after major design decisions are made, and those details are not typically permitted to lead to major design changes, as major changes can flow through to and invalidate other details. For example, in level design, mission and environment design details follow big brush-stroke design moves (in level design this is often called "greyboxing"). Progression design comes first – which determines which subset of gameplay is allowed to be used in a given level - then low-level details. This one-way dependency relationship between the larger shapes in gameplay and the details seems unavoidable.

As far as my experience went, I found it difficult to get my starting point for a new game concept within tools; trying to ideate new ideas by modelling ideas in an abstract language did not prove effective for me. I had more success translating concrete gameplay into abstract systems in order to trouble-shoot problems and balance systems in gameplay that I had already begun to design. This is no doubt in part because I do not yet possess sufficient literacy to imagine gameplay from abstract patterns and recognise what is "fun" purely within that abstraction.

7.4.1.1. Observation 2: Shifting focus within the representation

One of the drawbacks of representing at a low level of abstraction that, (as alluded to above in relation to the goal of a share language), is that it can demand a higher level of detail to represent a given element of a design situation than would a higher level of abstraction. At high levels of detail the readability of the visualisation can break down, with larger patterns and shapes crowded out by details - a sort of "can't see the forest for the trees" effect. Figure t shows an example of this in a *Machinations* diagram I made for a mode in *South Sea Trouble*.

Of course, as I discussed in Chapter 5, this is the problem of formal models – for the sake of computation, the model typically subordinates readability for accuracy.
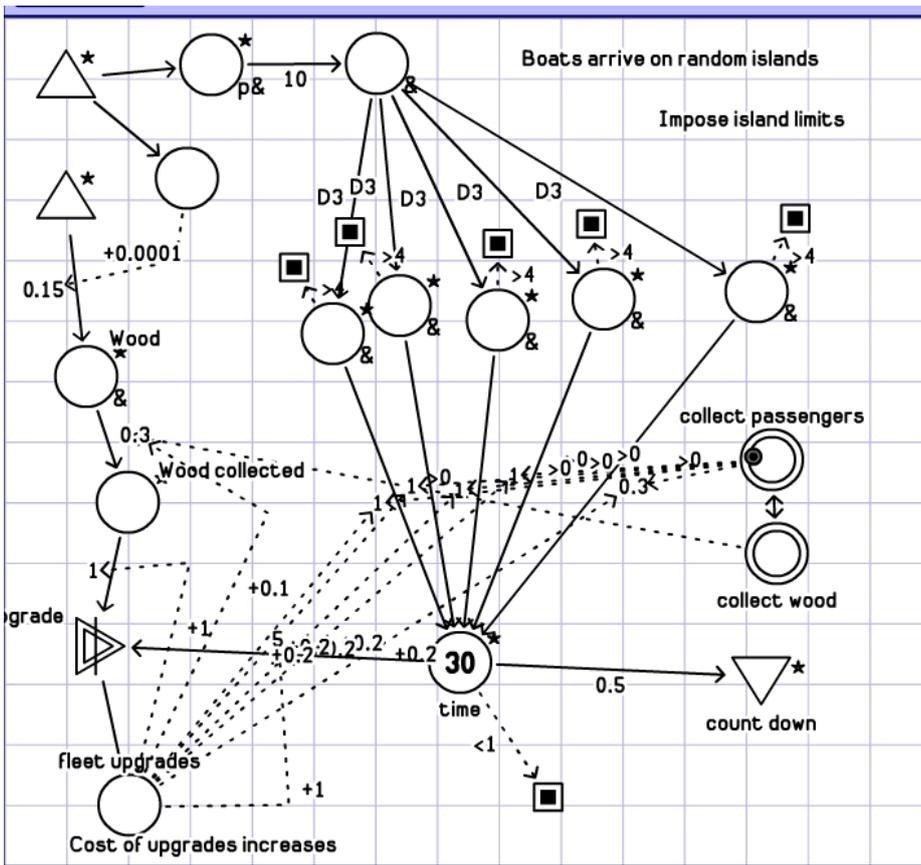
Figure t: A Machinations diagram containing so much detail that readability becomes an issue

A tool like *Machinations*, I have argued, could be thought to offer two forms of representation of the game design situation: *model* and *dynamics*. Viewed in this way, the way we might decide that the representation depends on their relative importance to the designer, as a function of context – which may well shift according to current design tasks or design progress. In other words, our design conversation may primarily be with materials that are the output of a simulation, and it is the simulation that we want talking back to us.

This scenario can easily be imagined at the end of a project, where design work can involve balancing and fine-tuning the data used by game mechanics rather than the mechanics themselves. At the game balancing stage, reading and analysing the second form of representation – the dynamics – is the priority. For the balancing task, accuracy and detail in the model is important. As a consequence, it is potentially baroquely detailed and unreadable. However, the designer may have less need of the first representation – the model of the mechanics – and

131

indeed may be treating modelling it as little more than a specification exercise where the model is already understood, and design moves are mostly minor tweaks to values in the model.

I used *Machinations* for this purpose to generate balancing data for *Ultraworm*. My attention was on the numerical data we input into the diagram and the peaks and valleys of the dynamics being rendered (i.e. using *Machinations'* charting feature) on a chart similar to the one shown at the bottom of Figure u.
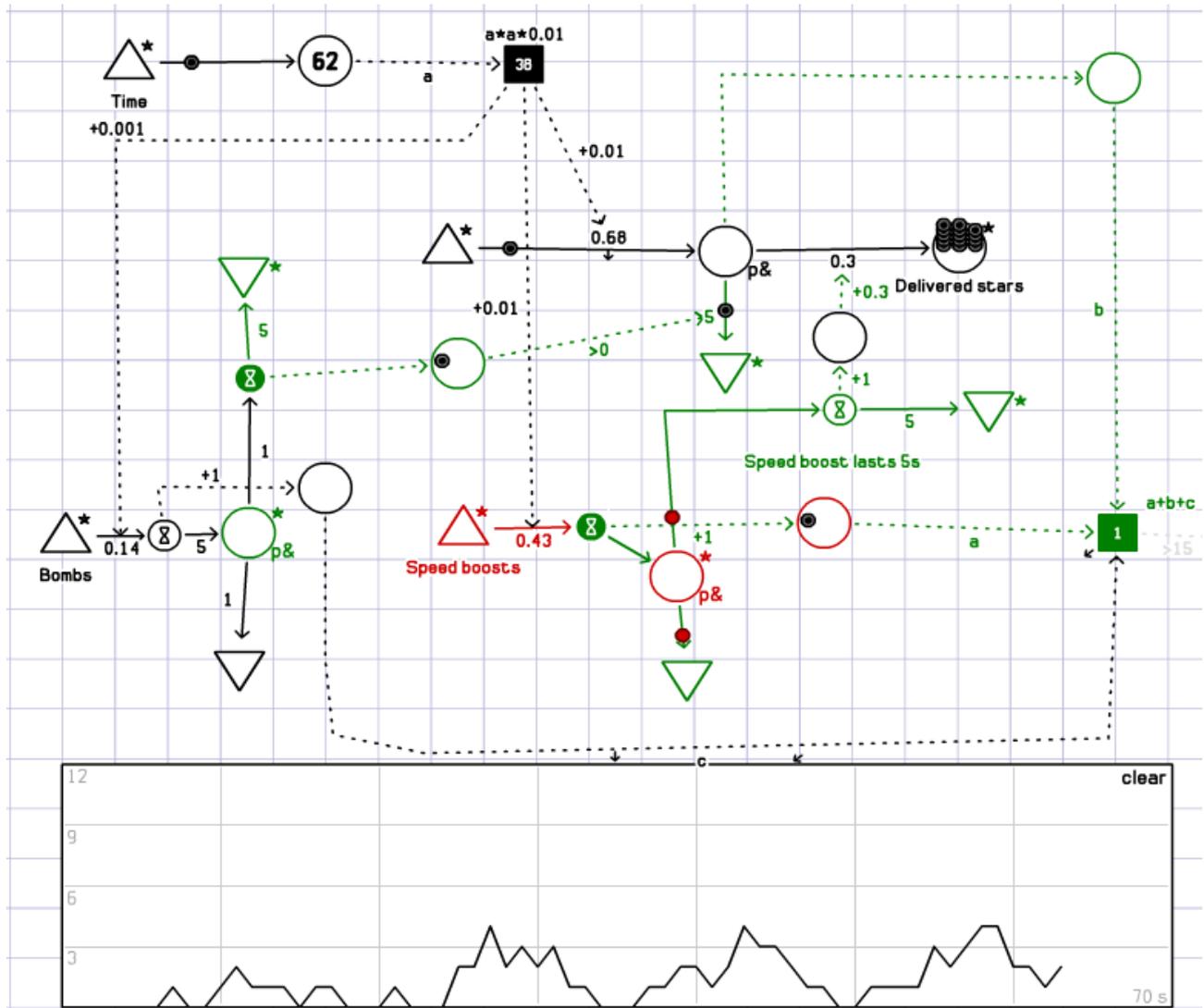

Figure u: Model of *Ultraworm* made in *Machinations* for the purpose of game balancing

Another way to deal with readability in the representation is to make sure that the designer is afforded control over the level of detail and abstraction themselves. One of the great benefits of *Machinations'* generic language is that it proves very flexible in terms of representing the design

situation at different abstraction levels. This flexibility helps solve the problem of readability as well, as a game can be modelled at a high level, then a number of diagrams can be produced that each detail subsystems of that design. Or within a single diagram, the designer can model the larger game system at a high level, then choose to "zoom in" on one particular subsystem and add low level detail to it. In addition, operating at different abstraction levels is useful to design thinking in and of itself. It affords the designer the ability to explore a design problem by working through the same problem at different levels of design detail, going deeper or higher to find the right level or levels at which they have the best view on the problem. Figure v shows how this can be done in *Machinations*: it shows a models of the same system modelled in Figure u but at a higher level of abstraction.
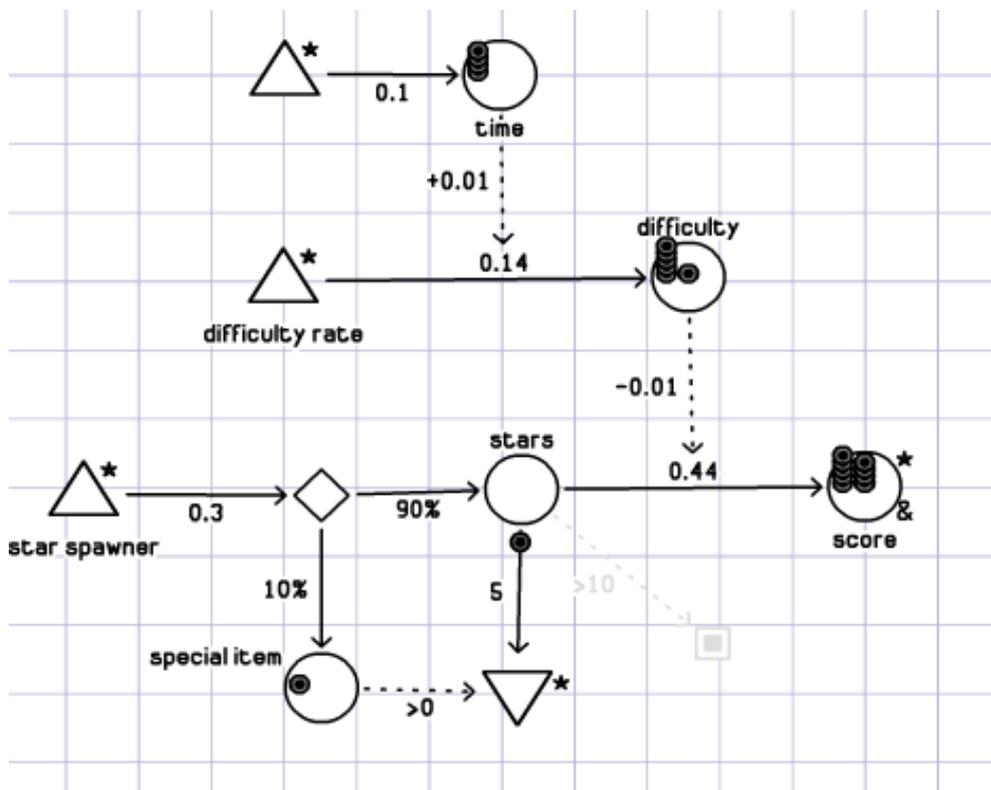

Figure v: A *Machinations* model of *Ultraworm* at a higher abstraction level than in Figure u.

7.4.1.2. Observation 3: Working on several layers concurrently thanks to PCG

These are abstraction levels within the narrow scope of game system mechanics, however. The problem described earlier in this section – the seemingly inevitable inflexibility in current practice where interdependencies prevent us from navigating abstraction levels freely – remain. With

Ludoscope and *Refraction's* tool, however, I saw how mixed initiative design can offer a powerful way to break this one-directional dependency relationship between abstraction layers. This is thanks to procedural content generation (PCG). PCG encourages the designer to design content in the form of rules or patterns, rather than as specific instances. By decoupling much of the work on low-level details from the use of those details in the game, major changes to the progression – as well as changes to the details themselves – can be made without as much of the detailed work being lost. In addition, having the high-level changes automatically flow through into the detailing in the levels allows the designer to quickly see what the effects of any major changes will be.

Refraction's tool – with its high level progression design tool integrated with its level editor – serves as an example of how this can work. As Figure w shows, *Refraction's* tool allows editing at multiple levels of abstraction – from the very high level of progression design to the low level of level editing. The abstractions exist concurrently, affording the designer freedom to make design moves throughout the design space at these different abstraction levels. When the designer makes a change in one component the tool automatically updates the other components.
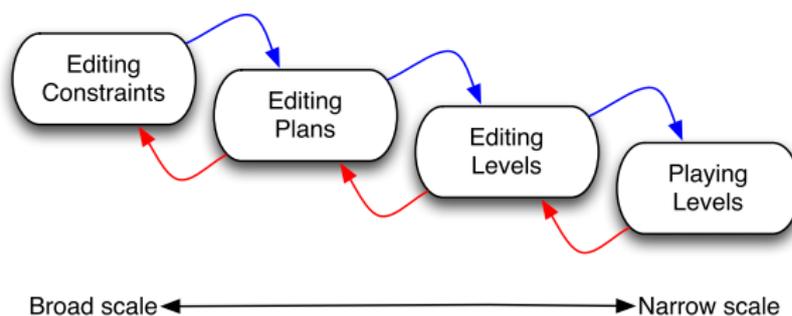

Figure w: *Refraction*'s tool's components enable design moves at a range abstraction levels

I took advantage of this approach for two projects: *The Casimir Effect* and *Loot the Room*. Using my tool based on *Refraction's* progression planning concept (see Chapter 10) to edit my game's level structure and progression rules, I generated level "recipe" scripts containing rules to direct the content generation for my levels using *Ludoscope*. This enabled me freedom to work in many areas of the design space concurrently, refining details and even some core mechanics, while also working on the "broader view" of game progression and level structures. While a final manual pass on each level was required, this felt like a small price to pay, as even a complete redesign at

the end would seem trivial compared to the laborious maintenance task of keeping the level content in sync with each change.

Figure x shows the different tool processes I used for *Loot the Room*, and how they fed back into each other. I was more or less able to work on all of these areas, or levels of abstraction, concurrently.
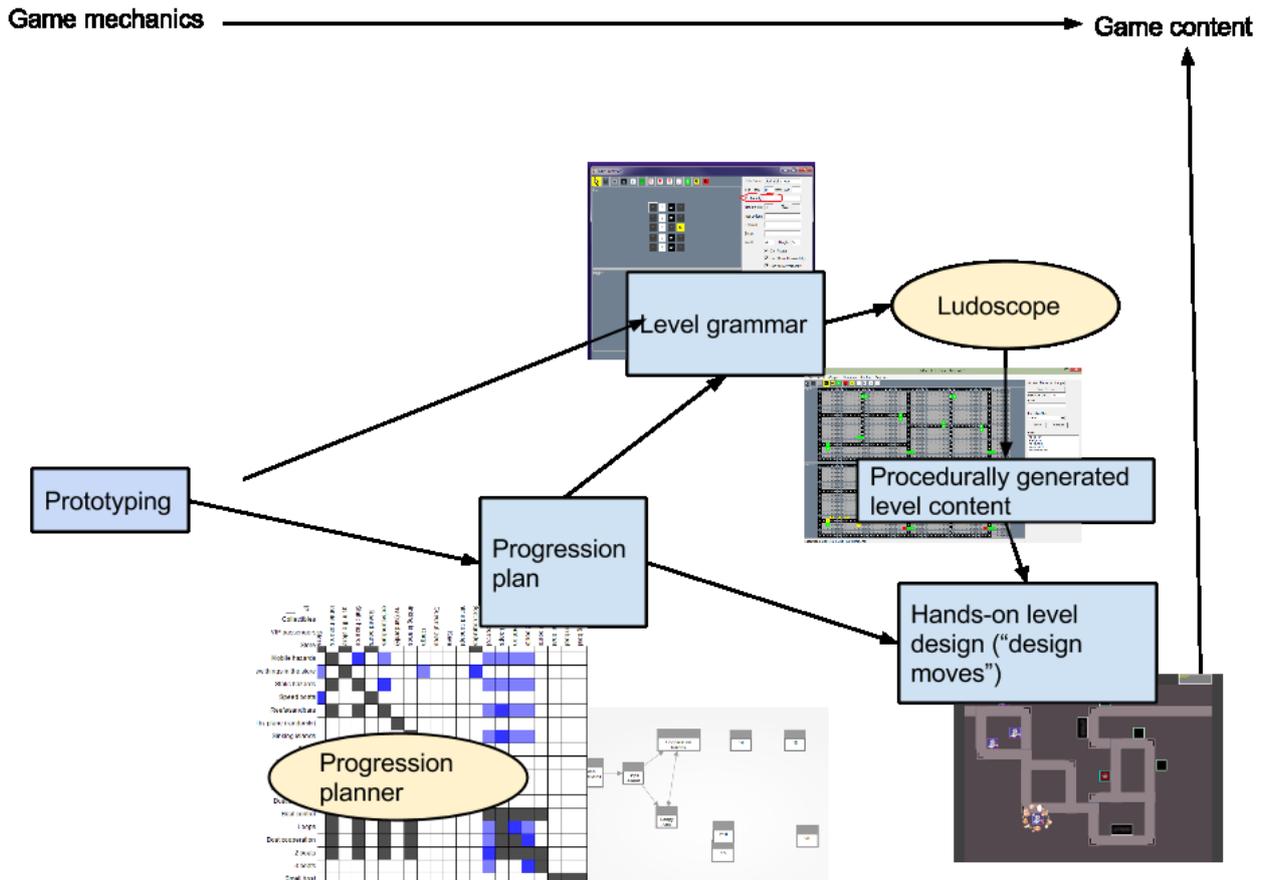
Figure x: Loot the Room design workflow

It is interesting to note that Dormans, creator of both *Machinations* and *Ludoscope*, in his personal practice prefers to work from layer to layer of abstraction as a stage-based process:

> I have chosen to focus on a particular order of design steps and transformations that in my experience is a sensible way of designing games. In this case, I propose that mechanics are designed before levels and when creating levels, missions are created before spaces.

Yet it is his tool *Ludoscope* that afforded me the freedom to take a rather more anarchic route through the hierarchy of the design space.

In addition to its content generation feature, *Ludoscope* is interesting in how it allows the designer flexibility in both the form and the level of abstraction via calculating model transformations. This approach allows a designer to work at the level of the concrete, while having the power to analyse their work and also manipulate it at a more abstract level, and vice versa.

7.4.1.3. High level sketching thanks to modularity

PCG is not the only way to be able to achieve rapid feedback – in other words, achieve a sketching-like process – for high-level design moves.

To allow designers to somewhat break out of this one-directional approach of level design described above, *Bethesda Game Studios* (the makers of the *Elder Scrolls* series of open-world RPGs) have devised a modular system for level building. It allows designers to work at a high level, using modular pre-designed chunks of level created by the team's artists. A new idea is not simply conceived as an individual element – it can be designed and added to the game globally in the form of a building block within a library.

This is similar to the modular approach used in architecture in the form of tools known as domain-oriented design environments. These offer built-in solutions that are commonly used and meaningful for a given design domain (Nelson and Mateas 2009). Nelson suggests that this approach could be useful in game design tools, enabling the designer to sketch at a high level of abstraction:

> …the direction proposed by domain-oriented design environments and especially metadesign [10] mapped well to some of the design issues we encountered. Game-design vocabulary is a mixture of existing terms inherited from previous games (e.g. RTS-design vocabulary) and novel ideas. Thus designers may want the ability to design higher-level abstractions than the state and state-evolution rules at which our tool (and actual game implementations) currently works, and to import existing representations where they exist. For example, our second case study would have found it useful to have his thinking prompted by a toolbox of off-the-shelf RTS design vocabulary (Nelson and Mateas 2009).

Applying the notion of domain-specificity at the level of game genres could also address the problem mentioned earlier related to the downside of having a shared language for games. This is,

to recapitulate, that in an attempt to create a universally-useful shared language for game, we risk reducing and complicating game elements so far away from their concrete form that the representation of the design situation is no longer useful. A vocabulary that contains domain-specific (in this case, genre-specific) building blocks could reintroduce elegance to such a shared language, hiding away the clutter of labour-intensive repetitive details to facilitate sketch-like design thinking at a high level of abstraction.

7.4.1.4. Observation 4: Nesting to facilitate work at a high level

An alternate approach to this problem is nesting. Nesting also can help the designer navigate the abstraction hierarchy and address the visualisation issues of high-detail representations is nesting. *Articy:Draft's* data flow view allows graph hierarchies, or "nested" graph structures: graph nodes can themselves contain sub-graphs, the nodes of which can in turn contain sub-graphs, and so on. Nesting is useful because it "clears the desk" for working at different levels of the hierarchy. On the down-side, however: 1) it creates hard, unambiguous hierarchies, as opposed to "soft", ambiguous relationships described above; 2) while hiding content "clears the desk" it also lessens the ease of retrieval and visual parsing; 3) the resulting hierarchy is not visualised as a whole in this format. This last negative is somewhat mitigated, however, by the alternative visualisation *Articy:Draft* provides in the form of a directory view window adjacent to the main window.

Figure y shows excerpts from two screenshots from an *Articy:Draft* project. The first shows a graph node; the second show the nested contents of the same graph node after the user has opened it.
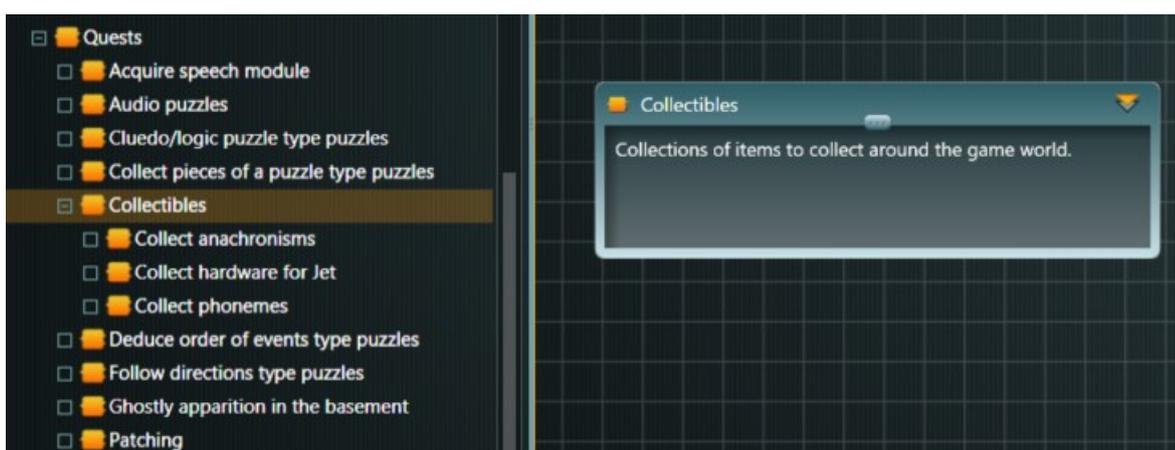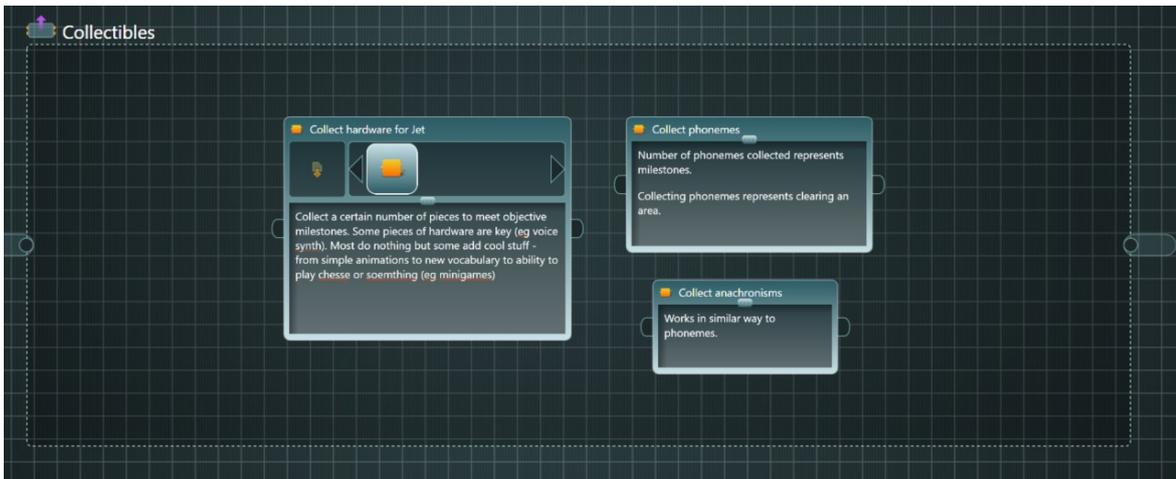
Figure y: Nesting in *Articy:Draft*

7.5. A more comprehensive representation?

Above I have discussed the importance of "game feel" and the difficulty of representing such high-level, aesthetic elements in a tool that endeavours to afford the designer the benefits of abstraction. This is an important problem, given that, as also previously discussed, such elements do not merely constitute a superficial layer of a design but can be fundamental to a game's design (user interaction and movement in virtual space in real-time action games being good examples).

In Chapter 4, I explained how a tool that represents game dynamics requires that the representation of the model be formal and, depending on the fidelity required from the representation of the dynamics, more or less comprehensive. But if all relevant elements cannot be represented in the model then both representations - i.e. the model and the dynamics are necessarily incomplete. Again, we return to the notion of a game design tool as a filter.

This is not necessarily a problem, however.

Schell comments that designers "seldom need to formally document the entire set of foundational rules (the underlying formal structure of the game) in a completely abstract way"(Schell 2008). We might conclude that they do not seek to represent a complete model of their game's structure in an abstract form because they lack adequate means with which to do so (hence the need for formal abstract design tools). Equally, however, it could it be that many elements of their games' structure do not need to be expressed beyond natural language for a designer to work on them.

138

There are often aspects of the mental image of the design situation in a game designer's mind that are not usefully externalised to a form any more formal than a text-based description. I have found that modelling some elements of a given game's design situation serves little purpose. This is not the same for all games; as we have seen, the location within the design space of important or challenging parts of a design can vary between games, and it can be hard to determine in advance.

This phenomenon of incomplete representation is not unique to game design. The more of the design situation represented, the more a designer might have to sacrifice the focus and control that a narrower view might afford them. It could also add to the labour required from the designer to use the tool, to the detriment of design thinking (discussed in Chapter 8). Do we even want to see all of the elements of the design situation all of the time? As discussed in Chapter 3, a designer feels a desire to control the complexity of a design situation in order to achieve focus. They do this, according to Schön and Lawson, by "identifying" a subset of important elements to be solved then "framing" – structuring the design situation accordingly. In this sense, the reduced, filtered representation a tool can enable could be a positive thing, as long as the designer has some control over that framing.

Finally, the goal of trying to achieve a comprehensive representation of the design situation may be impossible. In part, this could be because of the limits of externalisation and representation. Schön believes that many of the relevant variables of a design situation cannot be represented in a model (Winograd 1996). Schell comments that complete notation of games "might not even be possible" (Schell, 2008: 175). Perhaps, therefore, some elements of the game design situation will have to remain in the designer's mind or, at best, in natural-language based documentation until they are built into a playable version of the design (i.e. a prototype or the game itself).

7.6.   Design thinking adaption for tool representation

If we accept the idea that game design tools cannot be objective, neutral partners within a design relationship – that they introduce a new dimension of subjectivity into game design thinking, as well as a new layer of choices for the designer to make – the question then becomes one of what this means for the game designer.

To recapitulate: I have argued that while prototyping affords direct representation of the design situation, a design tool acts as a kind of filter through which the design situation is filtered. I have explained how the filter design of a tool is, essentially, a function of the choices the tool designer makes – choices about how the game design situation is represented, in terms of the form of abstraction (e.g. resources in *Machinations*), which kind of details are important (i.e. identification and framing), and the level(s) of abstraction.

Granted, many of these effects are not new to game designers; conventional methods of representing have this problem to some extent. The vertical slice-style prototype is biased too, in that it filters out the fun that may existing at a high level ("big picture") of abstraction. Such a prototype of a free-to-play game – e.g. *Cow Clicker*, for example, where a typical gameplay session involves clicking an image of a cow and the interest is rather in the long term progression and social aspects of the game – would have this problem. Conversely, written "high concept" design pitches tend to privilege game designs with exciting-sounding high-level concepts – a novel scenario or an innovative or unusual game mechanic – over game designs that have their USPs (Unique Selling Points) in small refinements to an established game genre.

The addition of tools into the design process, however, increases the complexity of this phenomenon: by offering more options, more choices to make. Tools impose a not insignificant additional layer of design thinking to the game design activity. The type of design thinking required contains some degree of Schön's reflection on action: thinking that not only in making some of these choices, but also recognising the particularities of the filter being used – essentially, a biased one – through which they are representing and viewing the design situation. Finally, as mentioned above in terms of representing in an abstract form, the quality of some of the choices made are coloured by the level of game systems literacy of the designer. While tools give us something, they also take.

Tool bias is, of course, not unique to game design tools. The model's filtering effect – conferring importance to some elements while excluding others – inevitably has an influence on the design thinking of the designer using the model. Some solutions are easier to generate with a given tool than others, as the design of a tool can privilege certain approaches and outcomes. Hence the common phenomenon of a "house style" being imparted onto the products designed using a given

tool. This problem is often seen in creative production tools, from graphic design to music software.

This additional design thinking, the reflection on action required of a designer using tools, is not unusual in other creative domains either. Modern composers and sound designers typically have not a single tool but a suite of software tools that they can call upon depending on the task, and their desired result. This is despite the fact that most modern audio software is feature rich, offering the designer the possibility of performing, at least in theory, all composition or sound design tasks. This benefits amateurs, who will often stick to using one tool. Expert practitioners, however, typically have a range of tools in their toolset, in order to have a richer palette of creative possibilities. They use their knowledge of the possibility space of each tool in their toolset, with all its limitations and biases, to select a tool that they feel suits given design context.

How much should this additional burden of design thinking matter for us in game design? While the problem of the filter is not unique to our domain, we might say that it is particularly relevant to game design tools. First, there is the comparatively large scope of the design space in game design, in part due to the large range of platforms through which games are mediated, from a playground to a deck of cards, a phone or a Virtual Reality headset; from single-player to massively multiplayer. Second, game design, typically a set of "second-order" non-visual problems, is a domain in which any representation of a game design situation, no matter how complete, is already a mediated and indirect representation of the game as a played experience.

If, as Schön says, unpredictability is a central attribute of design, game design is surely one of the design domains that could claim to be the best exemplar of that attribute. Our difficulty in "seeing" (i.e. within the "seeing-moving-seeing") impacts not only on our ability to confidently make design movies (reflect in action). I found that it even flows into decision-making about design tasks and tools (reflecting on action). It was often the case that I had to begin representing and solving the problem (analysis through synthesis) with a tool before I realised which form it would be most usefully dealt with in. This may be unsurprising, given that software developers, who also have trouble "seeing" in the face of so-called "wicked" problems (and for this reason often use iterative, analysis-through-synthesis methods) are prone to realising they have chosen the wrong tool or library for the job only once they have started it.

Why is this important? Because not understanding this can feel like failure – either on the designer's part, or on the part of the tool. To some extent, in my case, the tool's limitations and my personal limitations were no doubt to blame. And yet failure to represent a design problem within a tool, and failure to know whether the tool can usefully represent the problem anyway, are to some degree an unavoidable part of design. The fact that these repeated failures and the evidence of struggle is laid bare, in graphical form for the designer to see – instead of lying dormant within design documentation, hiding behind text – can be dispiriting.

It could be helpful, therefore, to liken the use of these tools to being fitted for glasses by an optometrist. While the optometrist can roughly estimate your prescription based on an eye test, the final test is a process of trial and error, in which the optometrist asks you to look through a series of lenses to find the one through which you can see most clearly. This trial and error process does not mean that the optometrist is bad at their job – it just means that some parts of the process are irresolvable problems that necessitate trial and error.

7.6.1. Observation 5: Documentation, sketches and prototypes to fill gaps in the representation

One of the benefits of creating an external representation of the design situation is that it can provide the designer with a structure within which to add any new ideas for additions and changes to the design. It can be a place for organising and ordering stray ideas.

However, as mentioned above, many of the relevant variables of a design situation cannot be represented in a model (Winograd 1996). Specifically, there will be ideas that cannot be usefully represented using the tool's given model. Having a place to store ideas that do not find a place within the representation is valuable (Hewett 2005). I found that being able to add annotations to *Machinations* diagrams for this purpose was useful. Some degree of natural-language-based game design documentation external to the tool seems unavoidable as well. For this I used text-based notes that I stored within a database. This was not always an ideal solution for my purposes, as it is a less than optimal means of organising volatile design ideas, for reasons already discussed. Later, I switched to using mind maps to store the same kinds of information but in a more visually scannable format. I also found some success in using *Articy:Draft* to store some of the more high level ideas (this is discussed in the following chapter).

Look and feel is something I also found myself at times needing to represent in a conventional way. In one instance I needed to imagine the moment-to-moment interactivity of one mechanic I was modelling in *Machinations*. I had previously the experience of modelling mechanics that seemed very good when I considered them in *Machinations* but turned out to be rather boring and repetitive to play at the level of interactivity within the game's environment and user interface. As discussed above, with my focus on the system modelling I found that I had a tendency to overlook other elements, feeling a bias towards the interest in the system dynamics I was modelling. This made me feel as if I was getting lost in the abstraction.

Filling this designer imagination gap with prototyping would be expensive, significantly slowing down the game sketching activity. A method I found that worked to address these limitations was to engage in a parallel process of rapid pen-and-paper sketching of mock-ups/wireframes, complementing the *Machinations* diagrams with my own diagrams showing gameplay interaction. These helped me imagine how my *Machinations* model might translate into moment-to-moment gameplay, by allowing me to think through the sequence and choices of actions the player would make as well as the movements of game elements within the visual game environment. Figure z and Figure aa show an example of this. The former is an interactive *Machinations* diagram of a game mode; the latter is a rapid sketch of how that game mode might play out on screen.

So, just as prototypes are often complemented with other materials such as concept art and documentation, we might need to take the same complementary approach to the use of design tools. First, to capture parts and details that cannot be usefully modelled. Second, to compensate for the abstraction of the design situation that tools necessitate. While abstraction is useful and necessary, it could have the effect of unbalancing our view away from look-and-feel dimensions of the design situation – dimensions that are not merely aesthetic but are to some degree interconnected with other elements in the design space.
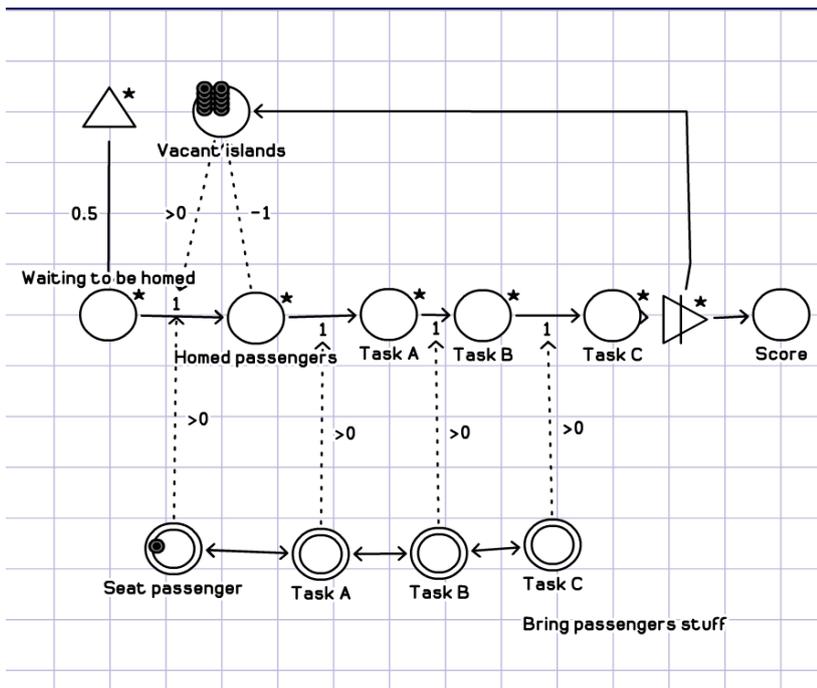
Figure z: A *Machinations* diagram of a game mode for *South Sea Trouble*
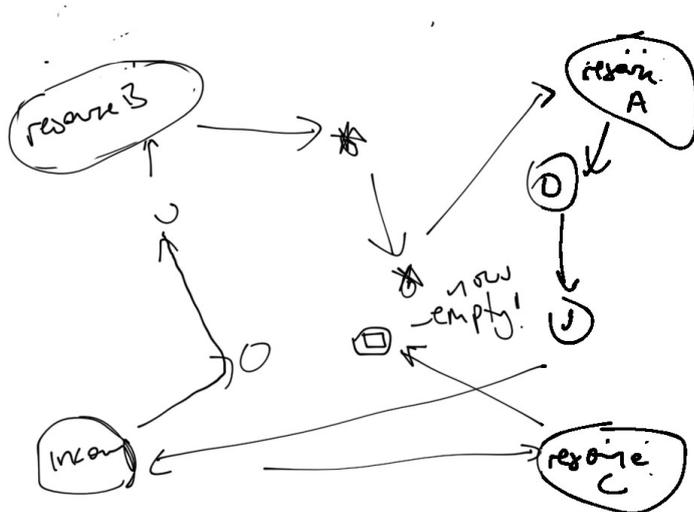

Figure aa: Rapid sketch of the same game mode on screen showing a sequence of game actions

8. OBSERVATION & ANALYSIS PART 2: THE ACT OF REPRESENTING

This chapter looks at the act of representing a design situation and making design moves with game design tools. I then go on to discuss the costs and conditions of representing and the designer's agency to make design moves. Finally, this chapter addresses the act of using the representation, to identify problems and possibilities.

In this chapter I offer three observations: how virtual space helped me organise ideas, the usefulness of an annotation layer, and how I found myself deploying content that was procedurally generated.

8.1. Tool labour

One of the practical considerations for using a tool is whether the benefits outweigh the costs in terms of investment required from the designer by the tool. This investment can be in the form of time (the "extensity of labour" [22] expended, as a political economist might call it) or mental effort (the intensity of the labour: "cognitive resources applied or allocated to a given design problem solving methods as compared with others" (Ben Shneiderman et al. 2005)). In this study, I am comparing the costs and benefits of using tools for game design versus current, unsupported game design practice.

This section looks at the costs of these tools: the investment in terms of both types labour invested in the tool. Specifically, I look at the investment in terms of quality (intensity) of labour (i.e. cognitive resources, skill) and the quantity (extensity) of labour. This labour comes in the form of the investment required to learn the tool, as well as the ongoing investment in using it.

It is important to note, however, that depending on the aim of a design tool, its goal may not be to increase the ease and efficiency of design tasks, but instead to afford the designer more creative possibilities. This attribute is known as a "high ceiling" and is discussed further on in this chapter.

---

[22]The amount of labor of unvarying intensity that a worker expends in the production process over a given time.

8.1.1. Accessibility threshold

A tool requires a certain degree of up-front investment from the user in the form of time and effort to acquire the skills necessary to use it effectively. In the literature, this is called a tool's "threshold" of accessibility – a low threshold requiring relatively less time and effort to learn (and remember what has been learnt) than a high threshold (Hewett et al. 2005). Depending on the tool, this investment can vary, with a low investment being more desirable.

As I have discussed in Chapter 6, an understandable reluctance to invest time into learning a tool that confers unproven benefits has been hypothesised as a barrier to game design tool adoption (Dormans 2009). Compounding this, as I argued in Chapter 3, is that taking a first step to switch from craft-based methods to a supported method (i.e. incorporating design tools into practice) represents a significant adjustment in and of itself, regardless of how accessible the game design tool or model may be. And finally, the lack of a shared language for game design or accepted and taught conceptual models, obliges researchers who create computational forms of design support (i.e. software) to not only develop a tool but also to develop a new model or language that their system can be based on. This problem facing a tool design researcher extends to the learning demands on the users of such software-based design support tools: they must learn and adjust to both a new system and a new language.

In this study, I have experienced this designer's pain of adjustment first hand. My perception is that it has taken me a significant period of time to let go of old, ingrained practices (the instinct to launch into prototyping and documentation, for instance), to learn to externalise my ideas in more abstract ways, and finally, to feel confidence in new, unfamiliar processes. Added to this was the more pragmatic task of learning the tools themselves. As anticipated, my learning experience was impacted by the production state of the tools, which ranged from experimental work still under development (e.g. *Ludoscope*) to fully-fledged, well-documented products (e.g. *Articy:Draft*, *Machinations*).

Unsurprisingly, it was the tools that were more ambitious in offering a larger possibility space – in the form of a rich set of concepts, complexity and functionality – that required more investment to fully exploit. Dormans concedes that *Machinations* is a big learn (Dormans 2012). Based on my experience with *Machinations*, this is less due to the demands of learning the feature set of the

tool and more because of the game system knowledge required in combination with the need to be able to understand and express this knowledge via Dormans' graphical language. These factors are discussed further on.

In general I did not feel, however, that any of the tools were needlessly complex to learn or had too high a threshold, in relation to what they offered. Indeed, some of the skills and concepts one needs to learn feel valuable to know as a game designer in general. That said, my expectations are coloured by a background in formal music composition, where the threshold for learning tools and languages is relatively high.

In addition to the time and effort required to learn how to use the tool, there is the ongoing investment required to use the tool.

8.1.2. Quality of labour

One dimension of the investment that a tool requires from a designer is effort. The amount of cognitive resources demanded by a tool is considered important; when these resources are expended on using the tool, less remain for use towards the design activity itself:

> When people are stressed or concentrating too much effort on how to use the tools themselves, then they will have less cognitive resources left over for use on finding creative solutions to their tasks. (Resnick, Myers, and Nakakoji 2005)

The amount of cognitive resources required can be described as "cognitive load". A concept used in the field of cognitive psychology, cognitive load refers to the degree of mental effort being used in the working memory. Experiencing heavy cognitive load is undesirable for a user, as it is thought to interfere with the cognitive task at hand and make a user more prone to error (Sweller, van Merrienboer, and Paas 1998).

8.1.2.1. The gap between the abstract and the concrete

Aside from superficial usability issues, a key demand on the user from a game design tool is the effort and skill required to bridge the gap from the form that the representation of the design situation takes (an abstract graphical language, for example) and the resulting gameplay.

As discussed in the previous chapter, a shared language requires the abstraction of concrete concepts in order that they be universalised. This requires something of the designer: each design move involves an act of imagination. That is, the designer must translate a concrete rule or mechanic into an appropriate abstract counterpart using the tool's language (in the case of *Machinations*, a resource flow-based framework). Then, the designer needs to successfully use their imagination to translate the dynamics that result back into concrete gameplay.

In a medium known for its wide diversity of forms, could a shared language for game design require a level of universalisation and abstraction that renders the relationship between concrete problems and their abstract models too remote and strained? Conceivably, the danger is that this constant translation in the designer's mind from the concrete to the abstract and back again is cognitively over-taxing.

In a discussion of the goal to create formal models for game design, Salen and Zimmerman allude to this tension between abstract and concrete thinking:

> A major challenge in creating a game design model is to conceptualize games on an abstract level, while also providing more specific rubrics for solving concrete game design problems (Katie Salen and Zimmerman 2005: 54)

I noticed that the mental effort required of me does indeed vary based on how much I need to fill the gap between the abstract and the concrete using my imagination. I found it to be variable, even within the same tool. In *Machinations*, for example, the further the mechanics of the game idea I am modelling are from resembling the resource flow metaphor *Machinations* uses, the more effort (as well as skill) is required of me to translate the concept back and forth in my head between the abstract and the concrete. For example, a god game involving resource harvesting and trade would correspond strongly, while a twitch-based action game would correspond much less.

I also noticed this variable gap when I built playable prototypes of what I designed, to playtest the results of my designs. The interactive simulation that *Machinations* offers – i.e. the way it allows you to, in a sense, "play" the model you have built – more or less resembled its playable prototype form as a function of how strong the game concept corresponded with the resource flow

metaphor. The less correspondence there was, the less – or at least different - insight I managed to derive about what the playable experience of the game might be.

8.1.2.2. The requirement of design literacy

The designer's literacy with a tool is not simply the ability to fluently read the language of a tool like *Machinations* itself. It is not even being able to mentally translate, element by element, how a model translates into concrete gameplay. More fundamentally, it is a case of being literate in the patterns that can be formed, and how those abstractions translate into the gameplay. A musical composer, for example, does not calculate in their mind the results of every interval between individual tones; they see them in their mind's eye in groups that form patterns, which may in turn form components of larger patterns. Literacy is about seeing patterns in the larger shapes that units form, and then associating certain patterns with gameplay outcomes they translate to.

This is catered for by Dormans, who has made resources available in the form of a book (*Game Mechanics: Advanced Game Design* (Adams and Dormans 2012)) and a wiki[23]. The material not only guides the reader on how to use *Machinations*, but teaches them some of the game systems literacy required to use it.

8.1.2.3. The challenge of the second order problem

Are there hard limits to this literacy? How readable can a game design model be in view of the fact that game design, unlike comparable design domains, is a "second order problem" in which designers are designing systems, rather than the results of those systems? In a way though, a kind of system thinking arguably operates in the minds of other designers too.

Again, I use the highly formalised domain of music to illustrate this. While music is typically composed directly, thereby not requiring composers to imagine music as dynamics resulting from a set of rules they are designing, understanding music as a system (the tonal system of Western music, for example) allows them to recognise, in a very abstract way i.e. not even imagining the resulting music itself, the aesthetic results of a system choice. To give a very basic example of this: a composer knows, for instance, that a melody comprised of tones that conform to a minor scale

---

[23] See http://www.jorisdormans.nl/machinations/wiki/index.php

(which is essentially a rule-set) has a certain aesthetic character, without having to imagine the melody itself. It is this kind of knowledge that can be applied to the design of music as an interactive experience – i.e. a second order problem, like a game. Indeed, it is this system understanding of music that enables composers to create algorithmic music (a famous historical example of which is Mozart's "Dice Music"[24]). This kind of musical systems literacy is typically acquired after many years of study, however. If game designers were trained to a similar level of literacy in game systems, the question would be whether the second-order nature of game design presents a hard barrier behind which some aspects are simply unknowable, or whether, like Western classical music, it is a very high bar but one that is surmountable with a combination of extensive training and support systems.

8.1.2.4. A path to systems literacy through reading dynamics

Dormans tells us that in its initial iteration, *Machinations*, like the diagramming systems proposed by LeBlanc and Koster, was a graphical notation language. The fact we can now use it to create a simulation and visualise the dynamics produced by our model gives us a kind of path to literacy in the notation system.

I have noted previously that game design is not yet a "conscious design" discipline. As such, game designers are not educated as part of a design tradition. Reading mechanics, isolated from the dynamics they produce, is not the way we tend to work. As craft-based designers our understanding of mechanics is very mediated; we associate certain mechanics with desired dynamics less through deduction and abstract knowledge and more by way of experience, rules of thumb and analysis of existing games. What dynamics will be produced by new, original mechanics is something we find out using trial and error in the form of prototyping. This is a very long process by which to learn the connections between game mechanics and the played experience.

Machinations, in visualising these dynamics for us, brings the second-order problem of game design into first-order scope. In so doing it creates a bridge to literacy: it inserts a visualisation of dynamics between mechanics and the played experience. It provides a bridge in the sense that,

---

[24] See http://www.amaranthpublishing.com/MozartDiceGame.htm

while we may initially struggle to read mechanics and recognise "fun" patterns, we are more able to read and recognise patterns in dynamics. While we may not be yet comfortable with reading the language of mechanics and imagining how they translate into a game experience, we are now learning how to interpret abstract dynamics into game experience. We know dynamics can be read and used by game designers (for example, in charts similar to the one in **Error! Reference ource not found.**), as data-driven design approaches are training us to read representations game dynamics data gathered from analytics and user testing.

Therefore, perhaps the most "readable" aspect of the model for the uninitiated is *Machinations'* charting feature. As part of the simulation, *Machinations* also allows the user to "tap" the game state at any point in the model to be plotted in real time on a chart. Designers will be familiar with this in the form of analytics and gameplay traces. It allows one to track "results". Figure bb gives an example of this: a diagram with a chart at the bottom, showing the results of multiple runs of the simulation. This charting feature is useful if the designer has a desired result in mind, based on game state variables that the player might themselves use to gauge the state of play. For example, these could be resources like money or hit points, variance in which can indicate how dynamic (and thereby exciting) is the competition between the player and the system or competing player(s); the point at which a long term strategy overtakes a short term one; how long resources will last before they run out; comparing the results of strategies to detect a dominant strategy.
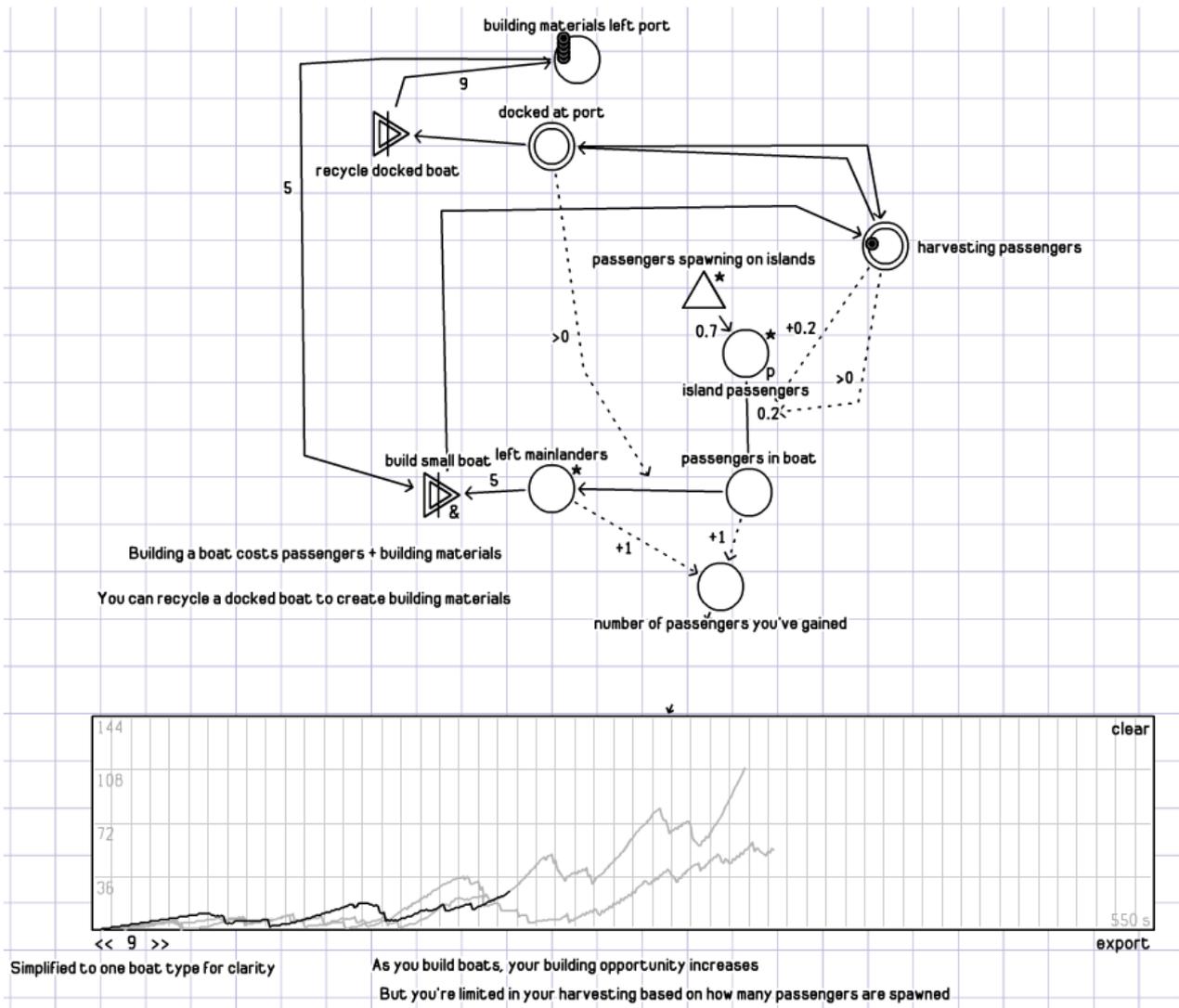
Figure bb: *Machinations* **diagram showing the system model above and the dynamics of the simulation in a chart below**

As for the readability of system models, the use of the machinations itself – the experience of comparing the playable prototype (or existing game) with its system model – can create the literacy required to make the tool useful. Composers, for example, read orchestral scores while listening to recordings, to see what orchestration techniques and instrumental combinations (akin to mechanics) produce certain sound colours (dynamics and aesthetics).

8.1.2.5. Comparison as an analysis tool helps compensate for lack of literacy

I mentioned above how one path to literacy is the analysis of existing games. While I often struggled to analyse my *Machinations* designs in isolation, I gained insight from modelling existing,

similar games and comparing them with models of my own design. This kind of comparison is something I am familiar with from conventional game design practice. Not only is describing our designs with reference to existing games a device commonly used in the absence of a standard game design vocabulary (Björk and Holopainen 2005), playtesting and analysing games to see how they solved a design problem similar to the one we are working on is a common design activity.

For example, in a mode I was designing for *South Sea Trouble* I was having trouble finding a way to incentivise the player to make sure all passengers were collected leaving none on screen. I was curious to see how *Diner Dash* – a very successful time-management game – did this. While I had a general idea of how *Diner Dash* worked based on having played it, modelling it in *Machinations* allowed me to identify two key mechanisms that nudged the player into doing what I needed my hypothetical player of my game mode to do.

8.1.3. Quantity of labour

Another dimension of the labour required from the designer is the quantity of labour in the form of time required: the amount of work required to produce results; the time it takes to make a design move. Measures and metrics suggested for the evaluation of creativity support tools typically include measuring time expended to devise design solutions (Hewett et al. 2005).

One of the key differences between "sketching" (i.e. design drawing using a tool) and prototyping is that sketching is, ideally, much quicker. Being able to make design moves at a speed that keeps pace with the speed of design thinking gives sketching a significant advantage for exploration of the design space. Specifically, a tool can enhance the sketching experience by offering quick results from hasty, only partially completed design work ("allow a partial effort to get a partial result quickly" (Resnick, Myers, and Nakakoji 2005)). This allowance for a partial, rather than a comprehensive, input of information has a meaningful impact on the design process.

In Chapter 2 I noted how in recent years the production of large, comprehensive game design documents has fallen out of favour. Their size (often hundreds of pages) made them not only time-consuming to write but also rendered information retrieval more time-consuming. One of the key problems was maintaining the document; as the design of a game rapidly evolves, keeping a large,

detailed document up-to-date can be so time-consuming and error-prone as to be counter-productive.

A tool that requires the user to input a large amount of information in order to get a result imposes upon the designer in a similar way. The "hunger" of the tool – the sheer quantity of time and information the tool demands from the designer – has an impact on not just representing the design situation (inputting/ building a lot of information) but also the design activities related to analysis (exploration of a large amount of information) and synthesis (making changes within a large information space). In this way, tool labour translates into the speed of the design process itself. Researchers have called this "viscosity" (Resnick, Myers, and Nakakoji 2005). A design support system with low viscosity, for example, allows the designer to make design moves - changes to the design situation – easily. Ideally, therefore, a tool aims to minimise the quantity of tool labour requirements on the user.

As we saw in the discussion of design process models in the previous chapters, not only is frequent iteration favoured in contemporary game design, it is a common feature of the design process in all domains; most design processes contain feedback loops (Dubberly 2004). The designer is constantly looping through the activities of analysis, synthesis and evaluation, each design potentially serving "as a springboard to a new round [i.e. iteration] of problem-solving" (Schön in (Winograd 1996)). The designer therefore benefits from being able to "see" the results of design moves easily and quickly, in order for this "seeing-moving-seeing" loop (Schön) of design to be as unobstructed as possible. Particularly in the earlier stages of design, the design situation is very volatile.

In the previous chapter I discussed how the cost of a comprehensive representation, or of working at a low abstraction level (requiring a high degree of detail to represent a design situation) can be an increase in the design labour required by the tool, and thus its viscosity. I also talked about my experience of how PCG enabled me to sketch ideas at a high level of abstraction – effectively lowering the viscosity of the design process.

8.2. Fostering the conditions for creative work

Jesse Schell, in his book *The Art of Game Design* (2008), highlights the importance of the psychological dimension of game design. A game designer is, in a way, collaborating with an unconscious version of themselves – one that is unpredictable, and easily frightened and must be treated with respect and delicacy. This emotional and personal dimension of any creative work is well known. It may be no surprise then that, particularly at a moment in history when game design as a creative practice is only just beginning to receive the cultural recognition it is due, the idea of adding computation and formalisation into the design process could be met with antipathy. With the game design community already feeling somewhat brutalised by the rise of data driven design, we see signs of designers railing against the idea of any kind of formalisation undermining the new, elevated status of game design as an art form in which we tell ourselves that great works of game design are the product of "divine inspiration"[25].

This idea that a designer in a creative domain must be careful not to "frighten the muse" extends to the tools that support creative tasks. In the domain of Creativity Support Tools, it is thought that "certain necessary conditions" must exist for support for creative work to be effective. Hewett talks about a tool providing at least necessary if not sufficient conditions for good creative work, based on what we know from psychology research. At minimum, a tool should avoid creating conditions for a designer that are known to "disrupt or to work against creativity" (Hewett 2005). Ideally, a creativity support tool should be "pleasurable and fun to use" (Resnick, Myers, and Nakakoji 2005).

The fact that tools for game design have not attracted interest by the game design community except on its margins might not be because the technology we have is not yet sophisticated or powerful enough; it could be that it is not yet simple[26] and gentle enough. Perhaps a more important assessment of a design tool is not the amount and quality of support it offers so much

---

[25] See Frank Lantz's essay "Against Design" for a recent example of this (Lantz 2015)
[26] Here we might recall how, as referenced in Chapter 6, Chris Hecker exhorted researchers to produce technology for developers that was as simple as "two sticks and a rock" (Hecker 2011)

as how well it manages to minimise any harmful side effects on the design process as a cost of that support.

8.2.1. The confidence conferred by constraints

While tools may have the potential to be brutal and demanding, one of the effects I experienced was the benefit that tools give in creating a safe space for creative exploration.

In Chapter 3 I gave an overview of the way design tools aid the designer in externalising their ideas by transferring the design situation in their heads into an external representation ("materials of design"). I explained how externalisation, in and of itself, is thought to have benefits to the designer: offloading the design situation frees up more space in the mind for design thinking.

Writing game design documentation does this. Documentation is passive, however; once you have unburdened yourself of an idea on one page, the word processor will not actively remind you of what you have previously said that may be relevant to what you are writing on the following page.

By contrast, mixed-initiative game design tools that offer the ability to check design moves against specified constraints, do take an active role in reminding the designer of previous design moves, helping the designer stay true to their design goals. *Refraction*'s tool's progression planning system of constraints does this, for example. By inputting progression rules one by one into the tool and assigning game concepts to levels, the designer is constraining the design space to a safe, secure space in which they are free to make design moves without fear. In a sense, this safe space has a function similar to Huizinga's "magic circle" (Huizinga, 1950: 10): the designer is inputting design rules to define the boundaries within which they, as a designer, feel free and confident to play. Creativity, like play, tends to thrive within known constraints.

Even simple computational support and visualisation can have a significant effect on the pace of design and the sense of confidence to make design moves. I found *Refraction*'s tool's method of constraint setting a very successful way to relieve the stress and cognitive burden of having to remember and adhere to my own, often extensive and interconnected sets of design rules that I would otherwise have had to halt work to reference from documentation and take time to calculate their relevance to individual design moves.

Added to this is the fear and worry a designer has that they have forgotten something, now buried in a document; or made a mistake somewhere. While we may feel nervous at the idea of letting computers anywhere near our creativity or getting in the way of our "divine inspiration" (to quote Lantz again) (ref), we do generally trust them for predictability, memory and basic computation. Recalling the design tool metaphors discussed in Chapter 5, the effect of externalising and representing material – even design rules – is like training up the nanny or collaborator I know I will come to need; when I am creating progression planning rules, for example, as if I were being the nanny or collaborator of my future self.

8.2.2. Ambiguity and imperfection

On the flip side of the security and freedom that constraints can confer, is the phenomenon of too much constraint. In a design tool, tolerance for ambiguity, inconsistency and incompleteness within a representation of a design situation is considered important.

Externalising the design situation compels a designer to articulate and concretise thoughts into a representation of their design ideas. Thoughts that were unformed must take some kind of shape in order to be represented, and they must be slotted in somehow into an existing structure. A tool treads a delicate line: while compelling the designer to formalise their thoughts it still needs to accommodate thoughts that might need to remain half-formed, or might contradict other design elements.

This is because design moves are exploratory – i.e. they are moves made by the designer for the purpose of "seeing". It is this very incompleteness, half-formedness and mess – the "ambiguity" that Lawson and others say is an essential creative thinking aid for the designer – that should be supported by a design tool. We do not want to be brutalised or bullied by tools; conversely, we do want to feel that they are robust enough to handle our imperfect, risky design moves. In other words, we do not want to feel paralysed by the uncomfortable possibility that if we make the "wrong" move we might make a "mistake" that breaks our design.

In providing the benefits of formalisation to the design process this structure and constraint is in tension with this requirement of tolerance for "ambiguity" in the representation. Does the rigidity of the visual language that we have reduced from the vague fluidity of natural language tolerate

this vital state of ambiguity that is core to the "sketching" process? Lawson refers to evidence showing how architects found they were unable to produce the same range of creative results with architectural CAD tools as compared to what they were able to do with pen-and-paper; he attributes this to CAD tools' intolerance for ambiguity.

Concretely, a tool can tolerate ambiguity by supporting the designer to maintain parallel or alternative solutions (Resnick, Myers, and Nakakoji 2005). *Refraction*'s tool achieves this by tolerating dissonance between the design rules that the designer sets for themselves and any design moves they make that break any of these rules. The tool informs the designer of the inconsistency but it does not constrain the designer from making the move and leaving the inconsistency in place. A kind of dissonance can be maintained, where design rules can sit alongside content that contradicts one or more rules. This contradiction can be resolved either in favour of the rule (the content changes) or the content (the rule eventually changes).

8.2.2.1. Observation 6: Tolerating design mess using virtual space

An incomplete design is a state of flux. It is not only the ideas themselves that are half-formed: connections between the ideas and their place within the design situation are not yet fixed. While still in this unresolved state, the designer must not only be able to externalise, but store, organise and retrieve their ideas. The affordance of logical and easily accessible places to externalise and store design ideas – especially incomplete, ambiguous ideas - that can be easily found later is extremely useful. Conventional documentation does not serve this purpose well. I found *Articy:Draft* to be a useful alternative.

Aside from plain documentation, a designer has other options for taking note of and storing their ideas, however. A conventional designer's notebook is an improvement on a monolithic document. Adding computational support, note-taking tools like *Evernote* use a tagging system to ease navigation and information retrieval. The content is still more or less obscured, however, and connections between these ideas (in note form) cannot be visualised, nor can they be presented together, for example, the way a writer organises scenes or fragments of a story as notes (on recipe cards, conventionally) that they can arrange on a surface like a table or a whiteboard in order to gain a holistic view. Figure cc shows a screenwriter engaged in this kind of activity.

Figure cc: A screenwriter viewing his notes laid out on a table (Kadoudi 2014)

Virtual space – taking advantage of visual cognition, like a "memory palace" - is a powerful way of organising even intangible and abstract information for easier retrieval. *Scrivener* (Literature and Latte 2007), a tool currently popular with writers, uses analogues of direct manipulation of ideas in space by the designer. However, while *Scrivener*'s corkboard offers the user a two dimensional space, the arrangement of cards is constrained to grid-based sequential format where ideas can only be reshuffled rather than placed and arranged freely in space. This sequential form is natural for linear media, but it is not suitable for games.

By contrast, *Articy:Draft*'s "open" geography of the spaces (shown in Figure dd) allowed me to make loose connections and groupings with high visibility. The virtual space afforded me the ability to pool ideas belonging to the same category. For example, I had a "board" (a screen) for my adventure game quests. A 2D space, as opposed to a directory structure, allowed me to make loose groupings, sub-groupings and "implied" connections and associations before deciding whether to make firm links between or categorisations of my ideas. Another way I was able to make connections that were loose and incomplete was using their flow fragment's entity referencing feature. Using this I was able to make loose associations between narrative ideas and quest ideas.
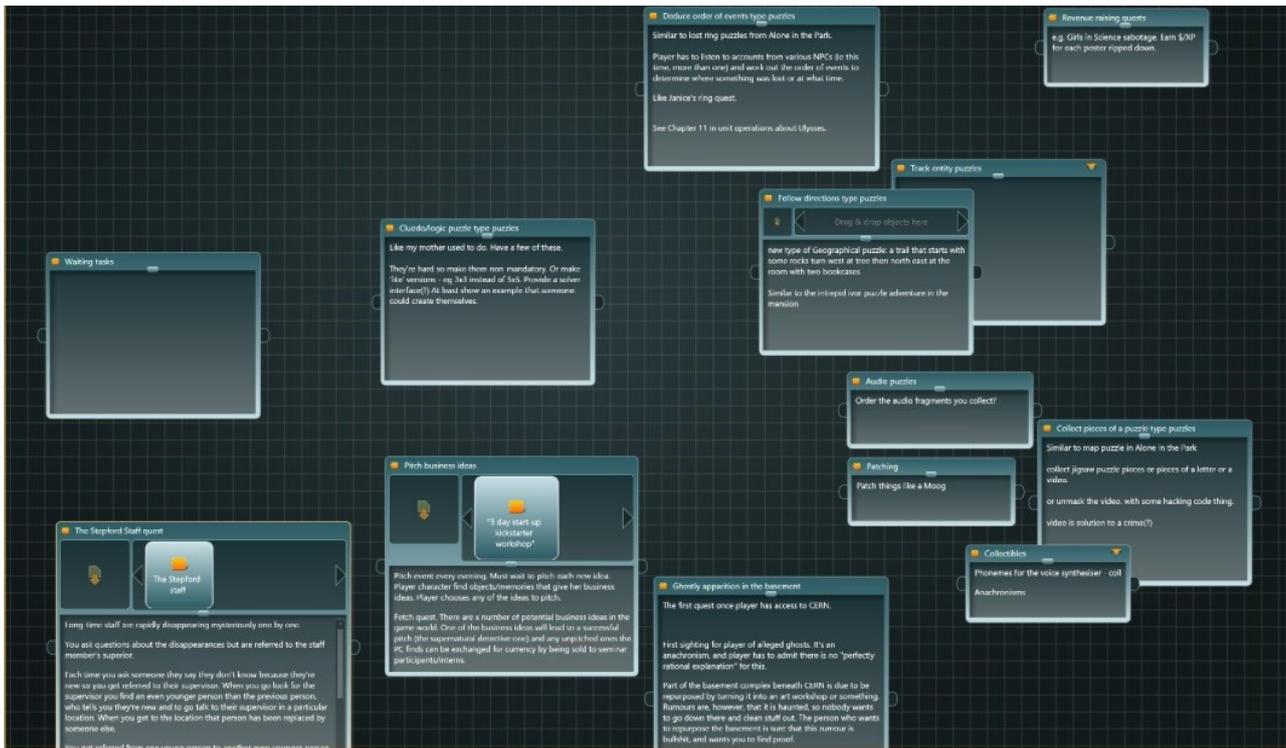
Figure dd: Loosely associated quest ideas in progress

8.2.2.2. Observation 7: Including an annotation layer

Hewett suggests that creativity support tools can provide "the electronic equivalent of sticky notes", in order to satisfy "the need to capture fragments of thoughts, ideas and relationships with minimal disruption of an ongoing activity" (Hewett 2005).

Unsurprisingly therefore, I have found that being able to annotate a diagram with text is useful. First, it helps with the readability of the representation. Annotation can also help with work in progress, reminding the designer of what is still to be done. Finally, as I have previously discussed, there are often aspects of the design situation that cannot be usefully modelled in a tool.

I made extensive use of such annotation (or text label) features in *Machinations* and in *Articy:Draft*. In Ludoscope it was not possible to annotate models (as models can be transformed into other forms, I can understand why), but I felt as if I would have liked to, in order to help close the abstract-to-concrete imagination gap while thinking through the structure of missions. While it is true that one thinks through these structures in a mostly abstract way at the stage of building mission graphs, concrete ideas (e.g. for using the abstractions) are frequently mixed in with these

thoughts. The concrete ideas can help form anchors for imagining how the structures one is building might work. For this reason, these concrete thoughts are not usefully abstracted and included in the model; they are best added to an annotation layer above it.

These annotations could also be in the form of images. Images are sometimes particularly good for expressing very vague, as-yet unarticulated or half-formed, ideas. In most game studios you will see images used for design purposes – in the form of concept art, reference images or diagrams – pinned to walls or to whiteboards along with text. While I found *Articy:Draft*'s workbench-style interface useful to do this with free-floating text-boxes and diagram elements, I would have liked to be able to pin images in the same way.

8.2.3. Freedom to move and explore

When we consider a tool in terms of the functionality it provides, it is tempting to frame our understanding of design activities as design tasks and how it helps us perform them. But this understanding of design as a set of problem-solving activities is rejected by some, particularly in creativity research. Instead of performing specific tasks, they believe, the primary activity of a creativity support tool should be "exploration", where the tools "define a space to explore, not a collection of specific activities" (Resnick, Myers, and Nakakoji 2005). Game designers interviewed by Nelson and Mateas somewhat echoed this view: they were keen to know, particularly for the earlier stages of the design, how a tool could help them "explore a design space" (Nelson and Mateas 2009).

In addition, it is important that designers are afforded some flexibility in the manner in which they explore and make design moves. In contrast to scientific problems, design problems have no "correct" paths to a solution. Lawson tells us that design problems (unlike scientific problems) are interpreted subjectively, and design inevitably involves subjective judgements (Lawson 2006, 3rd revise:124). Lawson uses the term "guiding principles" to describe the personal set of beliefs about design and design practice that drive variations in these judgements (Lawson 2006, 3rd revise:159). A tool that supports many different types of design styles is considered advantageous in a creativity support tool (Resnick, Myers, and Nakakoji 2005). As Nelson and Mateas point out, this applies to game design; design styles often strongly influenced by a game designer's personal design practices. Design tools, they say, should cater for this subjectivity and variation in personal

style (Nelson and Mateas 2009). There number of ways a tool can do this and I discuss these below.

One way a tool can support personal design style, as well as facilitate the exploration of the design space, is to maximise the number of possible paths that can be taken through the design space. Researchers suggest that a tool "support many paths" through a problem i.e. maximise the variety of ways the tool can be used to explore a problem and make design moves.

Secondly, the tool can offer the designer multiple ways to perform the same task or express the same idea. (Resnick, Myers, and Nakakoji 2005). This variety of means could also conceivably be achieved by the designer having access to not one but a set of tools that each facilitate different problem viewing and solving approaches. As I have discussed previously, any given tool will, to some extent, favour certain design approaches over others.

Thirdly, as well as offering multiple ways to represent the design situation and perform design moves, a tool can offer multiple ways to visualise the representation, giving different views on the same material. Many of these features were discussed in Chapter 7. For example, one way of gaining a different perspective is to view the data at different abstraction levels. As discussed, *Refraction*'s tool offers this as a core part of its premise. *Ludoscope*, with its model transformation-based framework does something similar.

Finally, flexibility can be offered by affording the designer control over the framing of the design situation. Being able to "zoom in" on some parts of the design while hiding others aids focus. *Articy:Draft* achieves this via its nesting feature, which allows the designer to nest sub-diagrams within diagram nodes. *Machinations* does not yet have this feature, which conceivably could be a useful way of dealing with some of the more elaborate diagramming that users may find themselves engaging in (perhaps especially for certain tasks like game balancing, as discussed previously). Music and audio-based software that use a patcher-based signal-processing interface (similar to *Machinations*' resource flow approach), for example, make great use of nesting to aid readability and debugging (e.g. *Pure Data* (Puckette 1997)).

### 8.2.4. Diversity of solutions within wide walls

Another dimension to the freedom to explore and support personal design styles is the flex giving the tool "wide walls", where the tool offers a large, open possibility space (supports a "wide range of explorations") instead of locking the designer into presets and predefined components.

A large possibility space allows a wide range of outcomes driven by personal styles. In the context of evaluating creativity support tools, Resnick, Myers and Nakakoji consider the diversity of solutions produced when using the tool as a measure of its success: "If the creations are all similar to one another, we feel that something has gone wrong" (Resnick, Myers, and Nakakoji 2005).

8.2.4.1. Observation 8: Domain-specificity and experimental design

The design for *The Casimir Effect* features a somewhat adaptive narrative. Game narrative, as I indicated in the description of the design challenges for *The Casimir Effect*, is rarely adaptive. Modelling the structure in *Articy:Draft*, therefore, was effectively an exercise in seeing whether *Articy:Draft* could support the visualisation of a game narrative with an atypical structure.

Space graphs for each level showing story-changing routes

The player starts at 'S'. Their goal is to reach Apartment F.

Orange routes modify the story in the subsequent level by "connecting" apartments.

Apartment A

Level/moment 1:

Broken leg in room, reading paper

Connection between A and C

Broken leg in room, chatting to woman. She has brought telescope and books on astronomy.

NO connection between B and D

Level/moment 2:

Broken leg in room watching TV

Looking with telescope., shocked

Connection between A and C

Level/moment 3:

Man working in apartment again. No flowers.

Woman looking through telescope, man pointing out the window, calling the police

Man reading paper.

Connection between A and C

Level/moment 4:

Man watches TV

Man is not there.

Figure ee: Excerpt from a diagram of the narrative structure of *The Casimir Effect* made in *Microsoft Visio*

At first, I attempted to model the structure in *Microsoft Visio*. *Visio* affords a high degree of flexibility in terms of the kinds of the customisability of the appearance of nodes and node text, graph edges and the ability to add visual cues such as borders around groups of nodes and free-floating text and images. I produced a diagram with two views of the structure. This can be seen in Figure ee. On the left are four level (space) layouts showing target locations (apartments in a building). If the player clears a certain path (marked with an orange arrow) between the target locations a "connection" is made between the apartments which affects the narrative events that occur in the connected apartments.

I considered that this diagram was not clear enough. I showed it to my collaborator and he found it hard to understand how the structure worked based on the diagram. I wanted to resolve this by better visualising the dependencies and their relationship to the different room states. In addition, by using the diagram for design thinking, I wanted to be able to work out the spatial connections needed between the different levels.

I then attempted to model the structure in *Articy:Draft* but quickly gave up. In *Articy:Draft* I found I had only two ways to view the relationships in my diagram – with graph edges or by opening up individual nodes to view their properties. I needed to view dependencies in a cleaner way than the tangled mess of graph edges I had created, and I wanted to be able to add large, highly visible labels that would provide a mid-level layer of visualisation to help me interpret my diagram.

My collaborator and I, when we next needed to communicate the actual content, found ourselves using a very simple lay-out in a spreadsheet program (Figure ff). This perhaps did not show the nature of the structure (which I had, by now, communicated to my collaborator verbally) but it was enough of a visualisation of the structure to think through and iterate over the narrative content.

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Possible connections:	A + D	E + D	E + B	E + A	E + C	E + B
		B + C	B + D	A + B	D + C	
Apartment D: office worker man & dog	Man is playing a console game. His dog is sitting beside him on sofa with a ball in its mouth.	Man is shaving.	Man is eating a pizza	Finishing a shower, man reaches for his towel.	Man is reading a book. Dog wagging tail wanting to be played with.	Man playing console
		Strange light. While shaving, man is alarmed to see the boy ghost in the mirror.	Man is eating a pizza. The fridge opens and closes by itself	Finishing a shower, man reaches for his towel – but it isn't there. Boy ghost apparent through steam behind him, with the towel over his head	Man is reading a book, looking anxious. Dog is playfully wrestling a ball away from/or being thrown a ball by an invisible presence (it's the boy ghost).	Boy ghost is smashing and throwing things around. Guy scared, up against the wall.
	the sofa, also playing the game.					
					Man is reading a book. Girl from apartment C is playing with his dog (which is also playing with boy ghost if he's there).	
			n door, has dog in her arms, giving him back to man.			
						Empty apartment if man has met old woman.

Figure ff: Excerpt from a work-in-progress narrative diagram made in *Microsoft Excel*

166

8.2.4.2. The downside of wide walls: not enough domain-specificity, too much labour

The limits and constraints that I have just described encountering point to a perennial design tool problem: the inevitable compromise that has to be made somewhere in between design support and design freedom.

In Chapter 7 I raised some potential disadvantages associated with achieving the goal for a "shared language" – a game design support generic enough for general game design use. As compared with a narrower, more genre-specific tool or language, representing the game design situation is more complex and labour intensive, as such a language would be composed of elements that would need to be at a more primitive, lower abstraction level than domain-specific elements. I suggested that, given how widely variable games can be, the lowest common denominator may be too low to be useful – at least not useful for all games. Further, I described how my experience with these tools has led me to think that no one game design tool will be comprehensive enough – or indeed should be comprehensive enough – to be all things to all game design problems.

On the other hand, generic tools offer a large possibility space (i.e. "wide walls"). More domain-specific tools sacrifice this, to a greater or lesser extent. In the previous chapter I raised the fact that a design tool inevitably bears the characteristic of inherent "bias" that to some degree steers the designer towards (and even locks them into) particular kinds of solutions. It is probably safe to say that the more domain-specific the tool, the more this effect is intensified.

My difficulty in attempting an experimental narrative format in *Articy:Draft* illustrates this. It is very much a domain-specific diagramming environment; that is, it privileges the support of modelling structures typically seen in games (RPGs, adventure games, etc.). In doing so, it narrows down and restricts the possibility space of a more open diagramming environment (such as Visio, or, at the extreme end of freedom, pen and paper) to a subset of game-typical affordances. In this way, the possibility space begins to become a collection of building blocks and templates – with all the benefits that confers. One of the trade-offs, however, is the freedom lost by this restriction.

As with domain-specific tools in other design domains, this trade-off and tension seem difficult to avoid. Domain-specific plugins and extensions to a generic system are one solution. Another is to accept the trade-off and consider whether a good balance has been achieved between freedom

and domain-specific support. Certainly, as far as I can see, *Articy:Draft* seems to cater to the needs of narrative for most common game genres.

8.2.5. High ceilings

A tool with a "high ceiling" enables a designer to create "sophisticated, complete solutions" (Resnick, Myers, and Nakakoji 2005). This is one of the primary goals of game design tools. Tool designer Dormans, for example, hopes that:

> Using the right tools designers can shape emergent mechanics to produce progressive experiences, and by having a clear perspective on a game's internal economy and mechanics designers can structure levels that go beyond a structured learning curve. (J Dormans, 2012: 159-160)

The importance of raising the ceiling of game design is also the desire of many working in the area of procedural content generation (PCG).

Procedural content generation for game design could have any of the following goals: creating a higher ceiling for design – i.e. to help game designers create better (higher quality, more sophisticated, more creative or related metrics) games; to help game designers create games more efficiently (i.e. expend less time and fewer resources to make existing games); to assist novice designers to perform game design tasks that they would otherwise need training to undertake. This final goal represents a scenario that is outside the scope of this study.

The goal of the higher ceiling seems to be the one favoured by the research community. At a panel session entitled "PCG Evaluation and Validation"[27] at the fourth Workshop on Procedural Content Generation (PCG 2013), participants expressed the view that the primary aim for PCG should not be merely to enable game developers to produce game content at a faster rate, or with less skill and effort. Rather, the goal more valued among PCG researchers was to expand the creative possibilities for game developers and for the games they are able to produce. While the discussion was about content-generation technology rather than design support, this value system applies to tools as well.

---

[27] See http://pcg.fdg2013.org/program.html

In our cost-benefit analysis of game design tools we should consider that even if a design tool increases the labour involved in a problem-solving task rather than reducing it, this may be balanced against the benefits of the heightened ceiling the tool affords.

That said, the two goals – of labour saving and increased possibilities – are not unrelated. As we have seen, reducing some of the burden - both quantitatively and qualitatively - on designers can help towards creating conditions for the extension of design possibilities and serve as a powerful aid to creativity. The designer, afforded more mental space and a sense of safety, is better equipped to take advantage of whatever higher ceiling a tool may offer.

In the previous chapter I described how procedural content generation (powered by *Ludoscope*) reduced the viscosity of my design process to enable faster design moves at a high level of abstraction.

I sense that there is a limit to this high ceiling though: when the tool is making decisions and suggestions that are too advanced for the designer to understand. If the designer does not quite understand the cause-and-effect of their design moves, then this reduces design moves to little more than an exercise in trial-and-error. Using a tool that is more advanced than one's capabilities can be counterproductive. One can see this happen in other design domains when novices use tools designed for experts. Even experts have their limits, however.

Ironically, I began to experience this after I added a feature to my own tool (described in Chapter 10). It was my own algorithm and in theory, I knew exactly what each component of it did. As a user of the tool, in the moment of making design moves, however, it was too complex to grasp, despite visualisation refinements to make it clearer. I eventually removed this feature and replaced it with a less sophisticated but much simpler alternative that returned some of the thinking/computation task to the designer, allowing the designer to undertake the task in an inferior, human way, at their own pace. As a user, I found the design experience significantly better.

8.3. Using the representation

We know that analysis and synthesis – in some form or another – are the key activities of the design process. To understand how a tool supports these activities it is useful to understand the process within which they occur.

As we recall, this process has been described by Schön as a "conversation with the materials" (of design). The "materials of design" designates the material form of the representation of the design situation. A good representation of the design situation provides good "back talk", uncovering implicit, tacit, and emergent dimensions of design tasks that designers may not have considered without the aid of representation (Fischer 2004). The designer hopes that the materials will "talk" to them after each design move, revealing new knowledge about the design, identifying both 1) problems and 2) new possibilities.

This back talk occurs because a design situation is by nature highly interconnected; it is hard to isolate and disentangle a single design problem from others within the design space. Lawson frames the design process itself as a complex system, in so far as when one makes a move within a system, the consequences of that move are beyond those that were anticipated. In other words, every design move a designer makes has "side effects". To highlight the complex web of dependences contained within design problems, Lawson likens designing to devising a crossword: if one changes the letters of one word, several other words will need altering, which in turn necessitates even further changes (Lawson 2006, 3rd revise:60). Schön explains how these changes are necessary: when design moves are performed in order to solve current problems, these moves typically give rise to unanticipated further problems to be solved in another area of the design space (Winograd 1996). More broadly, making design moves tends to reveal new information about the design – both problems and possibilities - provoking further design moves.

Game designers are familiar with the practice of eliciting back talk to reveal problems: i.e. of how externalising the design situation helps "beat the bushes"[28], flushing out problems that are not yet perceived. Problems can become apparent in the course of writing design documentation, for example. Finding problems is also one of the major functions of prototyping, of course; one of the

---

[28] "Beating the bushes" is a metaphor that evokes the act of beating bushes to scare birds into the open to be shot by a hunter for sport.

goals of prototyping is to reveal technical and design problems that, once identified, can be scheduled and budgeted for within the development cycle. Nelson et al. call this "testing-type prototyping" (Nelson and Mateas 2009), where the designer is testing the rigour of their concept, hoping to reveal any problems that may lie hidden in their design.

Nelson also identifies another, more open-ended and exploratory style of prototyping ("exploratory prototyping") where the designer hopes to discover "what is interesting in the design space". In other words, the designer is hoping to identify new possibilities – the identification of new possibilities and solutions being another kind of back talk. This can occur because representing – i.e. actively reconstructing what is in the mind - should, according to Fischer, allow us to form new associations between concepts (Fischer 2004). These solution side effects of design activities have been called "creative leaps" - novel or creative solution candidates that emerge unexpectedly while working on the design situation (Cross 2006). Another term used to describe these is "serendipitous solutions" (Hewett et al. 2005).

We can expect, therefore, that while working with a game design tool a similar problem and possibility identification process would occur. Problems, not yet apparent, will arise while representing and making design moves within the tool. In addition, new ideas may arise in a disorderly, unpredictable and even inconvenient fashion. As side effects to the immediate design task being worked on, some of these solutions may not relevant to the task at hand but rather to a different part of the design situation.

8.3.1. Observation 9: Solutions generated via procedural content generation

The act of representing my designs using tools did seem to help me identify new design possibilities and instigate serendipitous solutions. Using *Ludoscope*, a tool that uses procedural content generation, however, this effect was amplified. This seems logical, given that the function of this technology is to generate content and to combine content in new ways. In addition, serendipity seems natural emerging from a process fuelled by randomness and recombination. These were not "creative leaps" in a cognitive sense, and they still required me as a designer to identify them, but they were connections made not by me, but by the tool.

I even found that the solutions I generated did not always come at the right place and the right time. In other words, the "ideas", while coming from an automated process, not a human designer, did not always come about within a suitable context within the design situation. Just as

ideas for one part of the design can emerge serendipitously from the subconscious workings of a designer's brain while that designer is working on another part, it seems that a tool can also produce good solutions at inopportune times. In other words, we might say these are side effects, produced not by conventional design thinking, but instead something we might call "tool thinking".

But why should this occur? We might expect these connections to be somewhat predictable, insofar as design moves made within such tools are "made to order" – i.e. design moves made by a tool are only "exploratory" within predefined limits, their execution conforming to design rules and constraints specified by the designer. To understand the unpredictably we may have to recall what Lawson said about unpredictability: that it is in part due to the fact that design problems are often not understood until after solutions have been attempted. In this sense, the commands given to a design tool by a designer inevitably contain "errors"; the serendipitous solutions are artefacts of these errors.

I began to notice that I was using three different strategies for dealing with "good" solutions that I generated procedurally. This meant three different outcomes for content. In the first scenario, I used the content for the purpose (outcome) for which it was generated. The other two scenarios involved repurposing the content for something else. I describe all three below:

1) The solution generated is suitable for use in the part of the design for which it was generated. It is directly incorporated into the design situation as level or mission content (for example). In other words, it is used in a single place, as a unique instance.

2) The solution may or may not be suitable for the use for which it was generated. Either way, it is a "serendipitous solution", in that it seems like a good candidate for becoming the basis of a new design rule, pattern or constraint; as such it is "reinvested" into the rule set driving the content generation. For example, when I was playtesting something I had generated in Ludoscope, occasionally I noticed that the tool had combined one of my level design rules to create combat or puzzle scenarios that I felt were novel and valuable (in the sense that they were "more than the sum of their parts"). Hence I considered them "serendipitous solutions" and created new patterns from them that I could apply more generally.

3) The solution is good, but it is not working well in the context for which it was generated. I store it somewhere to be used later in another part of the design situation.

Scenario 3 requires a way of storing ideas to be retrieved at an appropriate time in the future. This suggests a role for tools. The challenge is how to store the solutions in such a way so as they are easy to retrieve - or, ideally, proposed to the designer – when the designer has need of them.

Earlier in this chapter I touched upon the challenges of the storage and retrieval of text-based design ideas. For game content ideas in any form – images of level ideas generated via procedural content generation processes, for example – we have this same challenge of storage, organisation and retrieval. Unlike in the scenario in which side effect solutions are produced by conventional design thinking, however, we have a 'secret weapon' for this that we can use, afforded us by the kind of design formalisation encouraged by certain tools – the content generation rules of *Ludoscope*, and the progression design rules of *Refraction*, for example. It is that these rules can be repurposed as a framework for organising and retrieving ideas. In Chapter 10 I describe how, in response to this phenomenon of side effects I built and integrated a storage and retrieval solution into my workflow.

8.3.2. Finding problems

In Chapter 4 I made a distinction between the tools in this study relating to their representation of the design situation: tools that provide a means of representing the design situation and other tools which, in addition to this, provide computational support that transforms the representation in such a way as to reveal new information. Here I consider what this difference has meant in practice.

We know that in other design domains, simple representation (i.e. without computational support) can reveal problems. This is because, as I have discussed above, problems can be revealed as side effects of design moves. More fundamentally, we know that, in other domains at least, simple pen-and-paper sketching of design problems can allow us to "see" the problems (recalling Schön's "seeing-moving-seeing" loop model of the design process). This is supported by the notion that visual cognition is thought to aid understanding of non-visual design problems (also discussed previously).

In Chapter 4 I reviewed some of the attempts at creating diagramming languages based, at least in part, on this assumption. Koster, for example, suggests that just by looking at a diagram we might be able to determine whether it is too hard to learn (Koster 2005). That is, it is hoped that a game designer could extrapolate concrete notions about the player experience from a system diagram. The hope is that we can imagine the behaviours that emerge from complex systems without computational support. While this has proved effective in other design domains, is this possible in game design? Alternatively, does the "second order" nature of design problems that sets our domain apart from others mean that diagramming alone is largely inadequate for "seeing"?

One point to make on the positive side is that not all parts of a game design situation are complex systems. Unsurprisingly, I found that I was able to read progression design and narrative structures quite effectively using simple graphical representations. For these kinds of views into the design situation of a game, where we are diagramming the higher level, non-emergent structures of a game, perhaps very simple visualisation is enough – as it often is within diagrams, flow-charts, and spreadsheets created and used in current practice. However, this tells us nothing new.

Views into a situation that have us looking into the workings of a game system, however, are where the "second order" problem lies. Earlier in this chapter, I described the high level of literacy and interpretive considerations required to "read" a game system diagram that simulates game dynamics. I argued that *Machinations*, by adding simulation to its diagramming language, offers a path toward literacy, without which a designer would have no option but to acquire a knowledge of system patterns and their effects via a highly viscous (slow and cumbersome) cycle of diagramming and prototyping. I tend to think, therefore, that while interpreting a static game system diagram may be possible, it is not ideal – at least not yet. Interpreting system diagrams sets a very high literacy bar; as such it does not seem like the best first step for a culture of design that does not yet enjoy the formal literacy of a conscious design practice.

Yet it is not inconceivable that one day this kind of literacy might be achieved. Composers often use an instrument (often a keyboard instrument) as a simulation tool to help them hear the results of their design moves. This simulation is not strictly necessary, however. Musicians with many years of formal training in the Western musical tradition have the ability to imagine melodies, harmonies, sound colours and how these elements combine (an orchestral score has around thirty different instrumental parts, for example), based on a notated score alone. To an amateur musician this can seem like an almost implausible feat of imagination.

Similar feats of imagination may well be possible for game designers of the future. On the other hand, due to the nature of games as complex systems, game designers may never be able to attain such high levels of insight afforded by skill and literacy alone. Realistically, however, the question of whether purely static system diagrams are useful or not is mostly redundant. Unlike for the kind of composers I have described, the tools of our apprenticeship are computers. It is probably safe to say, therefore, that whether or not game designers will need computational support to understand their system diagrams, they will use it if it is available.

8.3.2.1. Simulation and other forms of computational support

Simulation, like the game it simulates, allows the designer to view some of the emergent qualities of their design. In Chapter 5 I characterised this as a kind of additional form of representation of the design situation (*dynamics*) for the designer to view. A simulation of the game, unlike the game itself, can afford the designer a view of the shapes that might emerge from these dynamics over a long time frame – to recall Nummenmaa description (Nummenmaa, Kuittinen, and Holopainen 2009), it gives us a "broader view". We may also recall, however, the way in which there exists a tension between the needs of building a useful simulation (completeness, detail and accuracy) and key aspects of the design activity (need for low viscosity, readability, tolerance for ambiguity).

To some extent, as I have discussed, this balance can be regulated by the designer as a function of the task at hand: while performing game balancing, for example, one might preference the needs of simulation and use a high level of detail and accuracy. For the design tasks that are not concerned with tweaking details of an already designed system, however, the cost of building and maintaining an accurate simulation feels too high. Until the design is finished, we know that, whatever our intentions may be, we cannot be sure that we will not have to make significant changes to the broader structure of our model based on discoveries we make while refining details. As Löwgren and Stolterman told us (recalling Chapter 3), a designer cannot be relied upon to solely focus on this specification layer of detail until the design is finished, as they naturally find themselves moving back and forth the between abstraction layers. We might recall how even in outmoded, linear "stage-based" AAA development contexts, the designer was permitted (in theory, at least) at the "alpha" stage (the first point at which the game was produced to the point where it was fully playable) to make major changes to the design.

This prompts me to consider whether we are better served by transferring our more precise (verification style) design questions - and thereby our need for accuracy and completeness - to other kinds of tools, or even other kinds of gameplay representation. By other kinds of tools, I mean tools operate at a late stage: design-related production tools, like automated testing tools that use the game code itself to produce an accurate simulation. By other inputs I mean data produced via playtesting data that can substitute or simulate some of the dynamics that are not being produced by an incomplete model. I discuss this in Chapter 11, where I argue that when we consider features for a tool to support design thinking, we should consider whether these design needs are better serviced by other kinds of tools within the game production workflow.

A limitation with simulation in a tool is that the necessary level of accuracy may not be feasible, depending on the system being designed. In Chapter 9 I talked about how a tool has a certain bias, how tools must necessarily reduce and abstract gameplay concepts down, and how this indirectness between the concrete and the abstract can vary as a function of how much a given game design resembles the style of abstraction.

It would seem to make sense, therefore, that if the goal of accuracy of simulation is counterproductive to the conditions of design and that a high degree of accuracy is unattainable anyway, we should content ourselves with representations that prioritise simplicity, even if this means compromising accuracy:

> Simplicity of the simulation system is important not only in terms of the effort required to build and maintain it, but seems to also be linked with better design outcomes (M. C. Yang in Nummenmaa et al)

But what if there were a way to enhance the accuracy of the simulation, while still maintaining simplicity in the representation?

In cases where we cannot, or we are unwilling to model some details we could, theoretically, use data to stand in and complete the simulation. In other words, use data to simulate the dynamics of the incomplete parts of the model. Supplementary dynamics would be read in as serial data. This approach of blending data derived from playtesting with modelling echoes methods used in so-called "Rational Design" (described in Chapter 2), where a technique of deriving game progression data by extrapolating from and interpolating between playtesting data is used.

This is not possible in *Machinations*, but I developed a habit of doing a very basic version of this. In *Machinations* I could abstract most elements of my designs to its resource-flow based model. There were sometimes, however, when I found elements that could not model to a high degree of accuracy. I got around this by using averages and estimates that I derived from playtesting and representing these in the form of a simple *"source"* node generating tokens at an average rate. It was, in effect, a simulation of what was already happening in my game, built so that I could imagine design changes.

Hypothetically, this average value could be replaced by a more accurate simulation in the manner proposed above: for example, instead of using constant values I could stream in a sequence of values (for example, read in via a variant of the source node type) that I could either prepare manually or mine from a playtest of my current game build.

8.3.3. Testing versus evaluation

In Chapter 5 I described software engineering style testing functions proposed for game design tools. None of the tools I practised with for this study contained such features. However, my general experiences with the tools made me think about evaluation type design activities work within game design tools.

It may be useful to make the distinction between testing as a design thinking activity, and verification-style testing that seeks to reveal specific (i.e. not systemic) problems in the design materials produced. Recall from my presentation of Design Studies concepts in Chapter 3 that evaluation is not a distinct stage in the design process, but rather part of an iterative design thinking loop, along with, and feeding back into, synthesis and analysis. We might consider leaving the notion of "validation and verification" out of design tools, and instead think more in terms of the design activity of "evaluation".

If we think to the design activities that currently stand in for game design tools – i.e. prototyping and playtesting, we can see this difference. The way game designers currently evaluate their designs is to playtest them:

> Is the game accomplishing its design goals?.... Are [players] having fun?... These questions can never be answered by writing a design document or crafting a set of game rules and materials. They can only be answered by way of play.(Salen and Zimmerman 2003)

While both playtesting and quality assurance testing (which includes finding not just technical bugs, but design bugs) involve playing the game, they performed for different reasons and at different design stages. Playtesting a game is an evaluation activity that looks for broad, even emotional judgements that expand beyond testing for errors in design logic or answering specifically articulated questions. Indeed, being made aware of trivial design errors and inconsistencies in a prototype can even be distracting and counter-productive. A designer is instead looking for answers to questions that are often very subtle, complex and hard to express – possibly even questions that they do not realise they are asking.

A design researcher might tell us this inexpressible quality of reflects the complex, wordless nature of "seeing" in a "seeing-moving-seeing" loop. In game design tools, the ideal might be to offer a designer a way to "see" in order to facilitate an evaluation type of design activity which harnessing the designer's human powers of subtlety and complexity.

8.3.4. Limits of interaction with simulation as a representation of play

One possible solution is to simulate playtesting. *Machinations* (and before it, tools such as *BIPED*) allows the designer to interact with the simulation. This allows the tool to represent the dynamics of the game.

A comparison with "play" however, is limited; as a representation of player experience, the success of interactive simulation is variable. Above I raised the issue of the variable degree of indirectness of the resemblance between concrete gameplay and its abstraction representation. This also affects the designer's "play"-like interaction with the simulation: As the relationship between the representation and the concrete becomes strained, "playing" the model would logically offer a less useful idea of the player experience.

In addition, we might recall this notion of the tool as a kind of filter, and that some elements of the design situation are filtered out of the representation. As discussed above, with my focus on the system modelling I found that I had a tendency to overlook other elements, feeling a bias towards the interest in the system dynamics I was modelling.

Finally, my experience seemed to bear out this difference between play and interactive simulation. In the previous chapter I described how I found that the difference between the experience of interacting with a system in the tool and my experience of playtesting the same system within a

prototype meant that in some instances I failed to identify key problems in the experience. If we recall, this meant I sometimes had "false positives"; sometimes modelling mechanics that seemed compelling when I considered them in *Machinations* but, as I discovered when building prototypes, turned out to be rather boring and repetitive to play within the interactivity context of the game's environment and user interface – to an extent where I found myself compensating for this by sketching out the interactions on paper.

This "faux fun" seems to be one unfortunate side effect of the ability to interact with the simulation. Ironically, this interaction can be fun in and of itself – its own kind of fun. Fun at the layer of system interaction could be a good indicator that the game itself, at the layer of player interaction, is fun – but not necessarily. Here the designer risks building a "mirage" of fun – fun that is real only in the representation, not real in the gameplay they are modelling.

Another worry is that in some cases, fun is counterproductive; when the designer attempts to make their design fun to "play" in the tool, they could risk ruining the fun in the game. This could be because, as I have previously explained, too much focus demanded from one aspect of a game (e.g. the mechanics) can crowd out or distract from the fun in another part of the game (e.g. exploration or interaction with the game world).

For example, when using *Machinations* I became aware of my tendency to favour the modelling of gameplay that was "fun" to play in *Machinations*, and to disfavour gameplay that was not. This is similar to the "tool bias" effect. In fact, I began to worry that I was becoming biased towards designing things that actually resemble *Machinations* itself – i.e. sources, drains and tokens.

An awareness of this effect, in combination with (and no doubt also related to) my level of literacy, affected my confidence in my own ability to create new game concepts from scratch within *Machinations*. I began not to trust what felt "fun" in *Machinations* alone, in the abstract, without the idea originating from somewhere else, and in a fairly concrete (i.e. as opposed to abstract) form.

8.3.5. "Active evaluation"

Given these qualities and limitations of interacting with a simulation it may be best not to imagine this interaction as a simulation of or validation of the experience. Instead of thinking of this activity as simulated "playing" we might think of it more as "exploring", a form of active

evaluation. That is, we conceive of this interaction within the framework of a conventional design activity.

We are familiar with the role of exploration in synthesis and analysis, in the sense of exploring the design space by making design moves. This is a different kind of exploration: exploration as an evaluation activity. As evaluation it is essentially still a "seeing" design activity. That is to say, the designer interacts with the simulation primarily as a designer, not so much imagining themselves as a player. In this sense, rather than a simulation of "playing", it is an unpacking the design situation step by step to reveal and explore its temporal and emergent dimensions. We can still use this interaction to imagine the player experience, but this requires us to compensate for limitations and inadequacies with our imaginations; interpretation and extrapolation is required. This insight into the player experience is still mediated.

9. OBSERVATION AND ANALYSIS PART 3: INTEGRATING GAME DESIGN TOOLS INTO PRACTICE

9.1. Overview

In this chapter I discuss the practice context for game design tools, and look at design tools from the point of view of how they integrate into existing and future game design practice.

First, I consider such tools from the point of view of existing processes and relationships in both game design and production. I then go on to discuss how game design tools work together. Finally, I look at the relationship between the tools and designer.

I describe two observations: describing my experience designing progression for *The Casimir Effect*, and how I managed the needs of both design thinking and asset production for *The Particle Who Knew Too Much* using *Articy:Draft*.

9.2. Introduction

One of the evaluation criteria discussed by design support researchers is the question of how a tool complements others in the designer's family of tools and techniques (Hewett et al. 2005). Echoing this for game design support, Khaled, Nelson and Barr say we should aim to develop design tools that can be integrated within a game designer's practice. If new approaches to design cannot be appropriated and integrated within existing contexts, they argue, uptake will be low and the tools will remain relegated to research contexts (Khaled, Nelson, and Barr 2013).

As discussed in Chapter 2, design work is performed during the whole project's development cycle, from concept development to a game's release (and, within the context of the games-as-service model, beyond). The work for designers ramps up and down at certain points, but continuing throughout. This may be in part due to our crafts-based design process, and may change if we move towards a conscious design process where design is somewhat decoupled from production. However, this is the current context that tools must work within. Game designers are engaged in a range of processes that include prototyping, documentation, design-related production tasks (level editing, for example), automated testing, playtesting, and analytics. Here I

draw some of these other processes into the discussion, to understand how tools might integrate and be reconciled with current design practice.

One caveat to make is that as I have been using these tools as a solo developer some of this is necessarily speculation; I am compelled to extrapolate from my experiences as a solo developer and connect them to my past and current experiences designing within small to large development teams.

9.3.   Designing during production

Production needs are highly visible, and by virtue of this, relatively easy to argue for. They are relatively easy to discover, quantify, articulate and document. Moreover, production progress, though not always linear, can be demonstrated. Producers and technical teams are accustomed to building in-house support technology for content creators that has a direct, easily understood and perceptible impact on getting a game produced. In my experience, requests for additions to production tools that would have a more indirect impact on production progress (e.g. an improvement that would help artists locate assets in a database more easily) can tend to be deprioritised and sidelined. This is understandable within an environment of constant pressure to show stakeholders concrete, usually audio-visual, evidence of progress. In this way production exudes an aura of pragmatic worthiness, if not shades of moral authority.

By contrast, design needs, as we have seen, are much less visible and hard to explain. In addition, demonstrating design progress, especially progress that is either not playable or does not visibly span many pages of documentation, can be hard.

In the design camp, our defensive strategy in the face of production can be to envelope the design activity in an aura of mysticism (recalling Lantz's use of the phrase "divine inspiration"). In this constructed narrative, designers are cast as valuable, irreplaceable artists or auteurs, with game design a mysterious, almost unknowable process driven by "talent", "x factors" and "inspiration". But design thinking, while difficult to articulate, is not unknowable. We should be able to advocate for design thinking needs on the same pragmatic basis as for production needs.

The other defensive strategy casts game designers as crafts-people. This narrative, which I have described in earlier chapters, sees designers gaining more control by engaging in production themselves as modern artisans, by taking up the tools of production (sometimes literally paper

craft materials) and becoming strong advocates of iterative development – essentially an attempt to reimagine the engineering practice of game development as 21st century craft.

In such a context, design thinking, which is neither mystical nor artisanal, is a "hard sell". This is the background that we need to keep in mind when making decisions about how design tools integrate into practice. As we are not as accustomed to thinking about the impact of our development decisions on design thinking as we might be, we need to take a vigilant, proactive approach, opening our practices and assumptions to be challenged or even compromised in favour of the needs of design thinking and their agents in the forms of design tools and methods. Some ideas for this are suggested in this chapter.

9.4. Workflows

Here I describe my design workflows. They were not planned – I let them evolve organically. As with many projects, they reflect a set of influences and circumstances including the nature of the game, the production constraints, the guiding principles of me, the designer (recalling Lawson), the scale of the projects, and so forth. While they may not be representative of other peoples' workflows, they are representative in the sense that they indicate how workflow configurations and combinations can vary. I refer to some of these diagrams within the course of discussion.
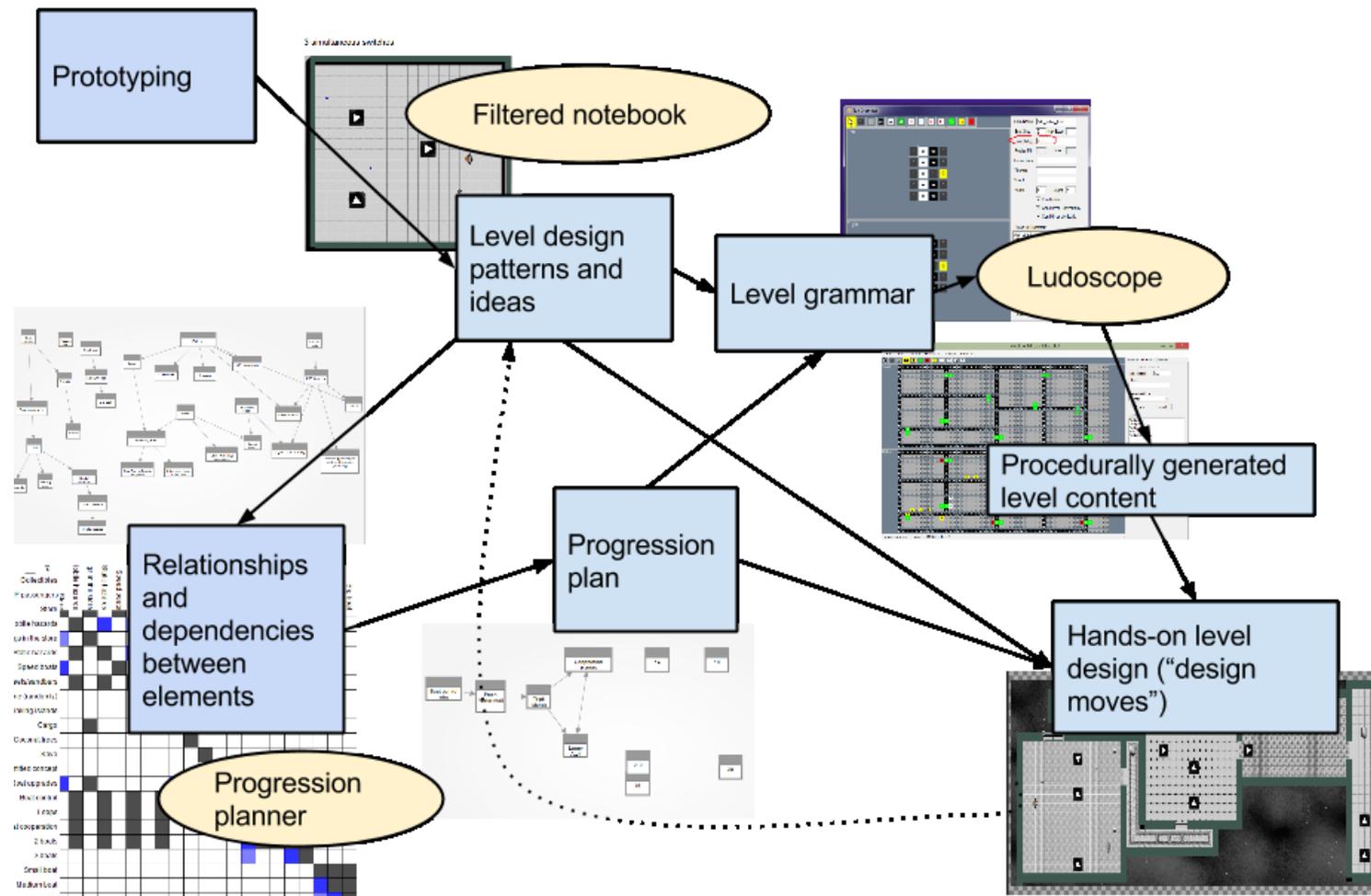
Figure gg: *The Casimir Effect* design workflow

Game mechanics → Game content

- Paper sketching
- Machinations
- Narrative progression design + mission ideas notebook
- Articy Draft
- Prototyping
- Progression plan
- Progression planner
- Hands-on level design ("design moves")
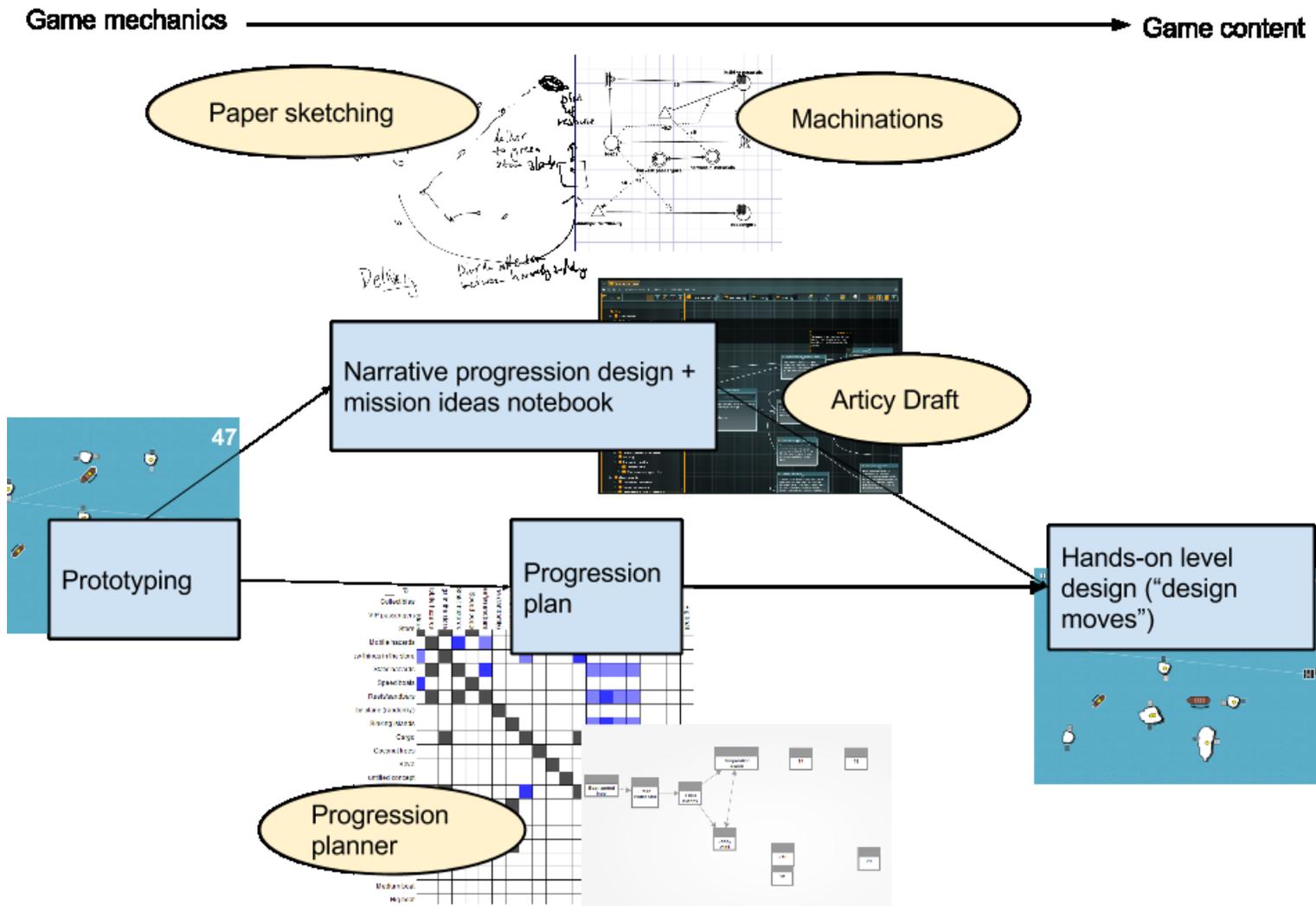
**Figure hh:** *South Sea Trouble* **design workflow**
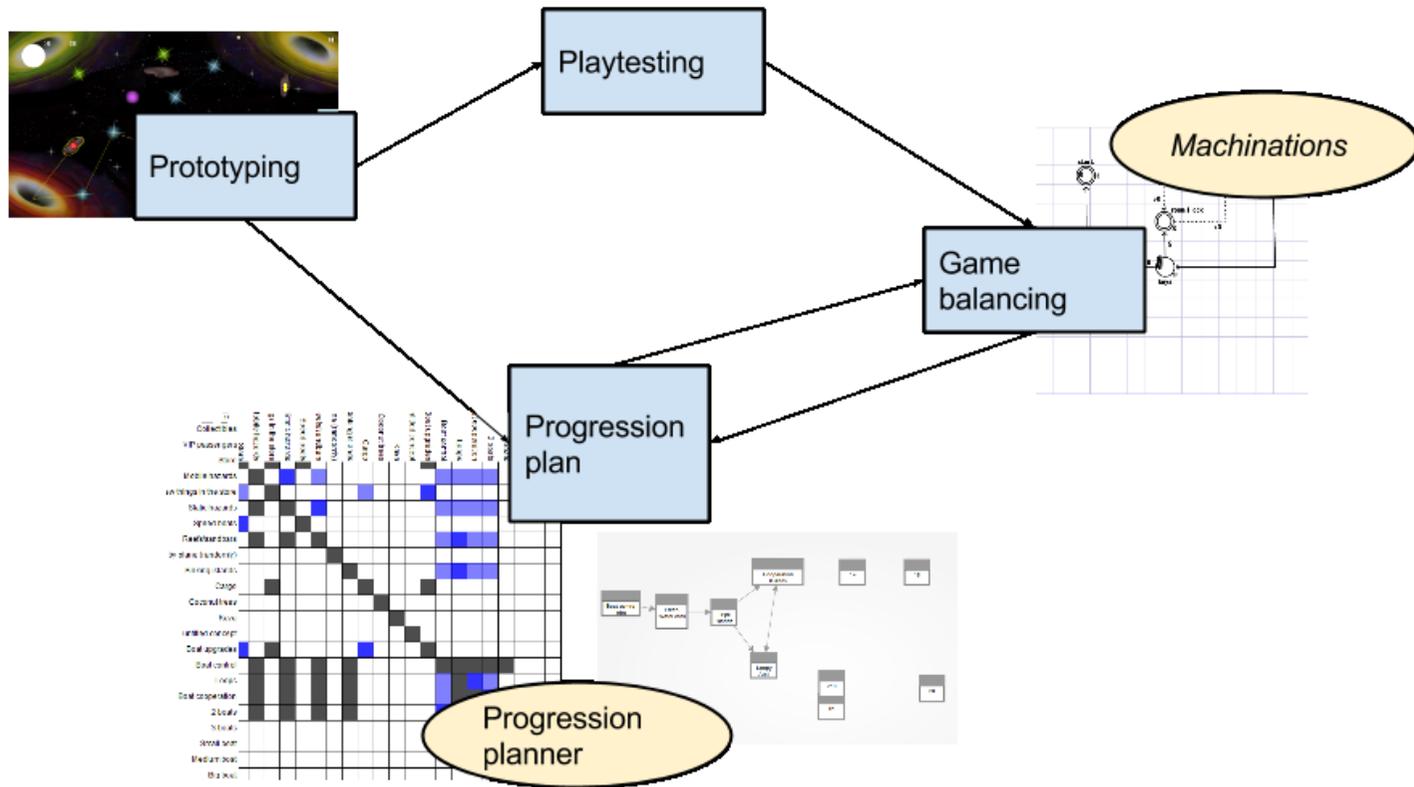
Figure ii: *Ultraworm* design workflow

187
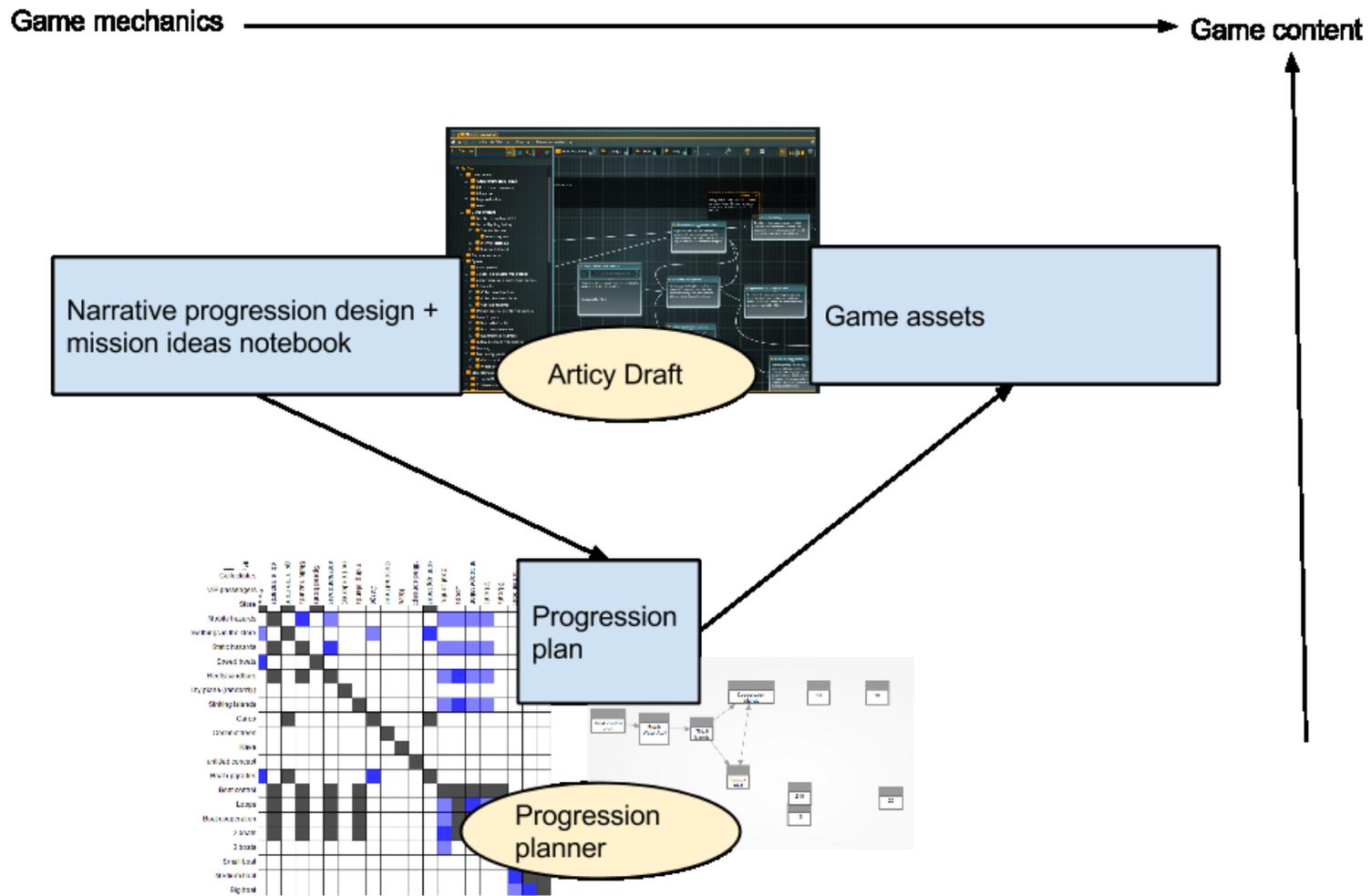
Figure jj: *The Particle Who Knew Too Much* **design workflow**

9.5.   A path from prototype to game (mechanics to progression) using tools

While prototyping should not be, as it is currently, more or less our only means of externalising the design situation, it remains an important tool within a game design toolset that probably cannot be replaced. As I have previously described, certain aspects of a game design situation – particularly "look and feel" – are not well served by abstract models. Vertical slice style prototypes using production tools are more suited for representing these aspects. Adam Smith suggests, for example, that his system *BIPED* "would be used after initial playtesting with a physical prototype (which gives the designer experience with the game's foundation using soft, negotiable rules)" (A. M. Smith, Nelson, and Mateas 2008).

For these reasons, perhaps, rapid prototyping is sometime also a good early step for generating new concepts when it comes to particular kinds of game ideas. My new ideas often come in the form of imagining concrete gameplay in my mind. While I have often brought these ideas directly into the design tools without prototyping, at other times I have found it useful to ideate, create and test new gameplay using prototyping. This is partly due to the fact that, as discussed previously, I have often found it difficult to ideate new gameplay ideas, and have confidence in those ideas, using tools. In addition, it is easy to generate and stumble upon new ideas in the messy realm of the concrete.

After prototyping ideas, I have then brought them into design tools to organise and understand them. For example, with *The Casimir Effect*, I used prototyping to feed ideas into skill chains and progression planning.

The design activities of prototyping and progression planning, in particular, feel very natural alongside each other.

First, prototyping provides a content pipeline of concrete gameplay from which to derive ideas: a small vertical slice of gameplay is a rich source of ideas from which to extrapolate patterns and grander structures, or new directions. A fitting metaphor might be one of panning for gold in a stream.

189

Second, progression planning frames an explorative prototyping process with larger goals. The designer is encouraged to analyse the slice of gameplay they have created, and formalise it and find a home for it within the broader context of the design situation as a whole. The designer asks: does this piece of gameplay represent something new (as compared to what is already in the design situation)? If so, give it a name and a place within the design situation (and within a tool, or a notebook), and relate it to other elements. Is it a variant? Finally, have I learnt something new about the design situation from this?

Hence, a process is created: new knowledge revealed by prototyping is fed into the more abstract representation of the design in a tool. It is a process that provides a kind of pathway from prototype to game. In this way, tools can provide or suggest a means and a process for turning a successful gameplay prototype into a game.

9.6.   Observation 10: From prototype to progression design via formalisation

As previously discussed, one of the tasks of a designer is to translate game design – the system and rules of the game – into level (i.e. progression) design. Here I describe my experience of how tools took me on the path from prototype to game, in the case of my project *The Casimir Effect*.

The Casimir Effect began as an idea for an original game mechanic that I diagrammed on paper. As it is a timing-based action mechanic, I built a small prototype to playtest it. Once I had confirmed that the idea was fun, I wanted to see whether I could generate enough gameplay out of this core mechanic to base a game around it. With this in mind, I extended the prototype, creating several combat and puzzle scenarios, and found that this was indeed possible.

9.6.1. Modularising gameplay into gameplay units

With the intention of showing it to other people to get their feedback on it, I then structured my prototype into the form of a gameplay sequence/linear level that in effect was a short tutorial. I did this in order that my playtesters could quickly learn the gameplay. In addition, I knew that I would be making a skill chain and would eventually be breaking down gameplay into units to be rearranged into a progression plan (i.e. using Butler et al's progression planning approach).

Hence, I spit my ideas into individually playable design patterns in the form of "micro-levels" – each demonstrating a unique unit of gameplay: an entity, action or puzzle type – which I could

easily order and re-sequence at will. Doing this required me to come up with a unique name for each micro-level. This was useful; having to distinguish, articulate and distil the character of each gameplay unit rendered my design materials clearer to me.

For this I felt no need of any formal process or tool. Later, as the gameplay grew, this linear sequence became unwieldy.

9.6.2. Skill chain diagramming on the path to progression

I reflected that, as Koster argues in his book *A Theory of Fun for Game Design* (Koster and Ebrary 2005), an entire game could be viewed as a process of skill acquisition and mastery. As an action game with puzzle elements, *The Casimir Effect* is very much a game of skill. I therefore decided to use the process of devising a skill chain as a first step in the game's progression design, shaping the game using this lens of skill acquisition.

In Chapter 5, I mention Cook's example of the skills acquired in the first several minutes of playing a classic side-scrolling platformer. In some modern side-scrolling platformers, however, the game may continue to introduce new enemy types, new weapons and special powers over the course of the game. One could imagine, therefore, that for certain genres, the skill chain concept could be usefully applied beyond the initial tutorial phase to help drive progression design for the entire game. Indeed, Cook proposes that "by linking more and more atoms in, you build a network that describes the entire game" (Cook 2007). This seemed applicable to action and puzzle genres, for example, where increasing gameplay complexity can be core to the game's progression - instead of, say, exploration or narrative. In a skill-based game, new gameplay generally means new skills the player must learn. *The Casimir Effect* fits into such a genre: a skill-based action game where progression is based on the addition of increasingly more advanced gameplay, based on the introduction of new variations to and combinations of a small set of game mechanics.

9.6.2.1. *What is a 'skill'?*

Through the process of creating a skill chain, I discovered that for this game it was unhelpful to make a distinction between the player's acquisition of a literal "skill" and the introduction of any other new type of game content. Modelling the skills separate from this other kind of content did not make sense. In *The Casimir Effect*, the player must discover, for example, that a particular combination of materials can be used as a weapon (specifically, a "bomb"). Acquiring the ability to use a bomb is a matter of being taught what this combination of materials looks like, and using a

set of action-based skills to then set up and trigger the bomb. In this way, the "skill" of using a bomb is effectively a composite of both knowledge and skill. In other words, it makes sense to treat "skill atoms" as a very broad concept that more or less means "types of gameplay". Indeed, Cook's skill chain concept was one of the inspirations for Butler's progression planning concept – in which Butler's term for types of gameplay is "game concepts". The experience made me think it useful that we think of gameplay units in very generic terms. I took this approach when designing my implementation of progression planning (described in Chapter 12).

9.6.2.2. Side effects

I repurposed my micro-levels as "skill atoms" and put them together into a skill chain diagram using *Microsoft Visio*. In doing this, I found I was also generating new content: in trying to build a comprehensive skill chain I occasionally discovered new skill atoms or combinations thereof that were not yet represented by micro-levels and found myself creating new micro-levels from them. In other words, analysing and formalising my materials had side effects in the form of generating new ideas.

9.6.2.3. Readability issues

From a starting point of a completely unstructured pool of gameplay atoms/micro-levels it was useful to begin to see the game take on some kind of shape in the form of this tree-like skill chain structure. As the design advanced I found that learning to play my game would require a great deal of skill acquisition and my skill chain became very large.

At this point I began to discover the negative side of using the lens of the skill chain to give shape to my entire game. At a certain point, from the point of view of a thinking tool, the visualisation benefits of seeing the "shape" of the content breaks down; the details visually converging into something shapeless and unreadable. It was an improvement on my unstructured pool of micro-levels, certainly, but the size and complexity of the structure rendered its use as a reference model to check design moves against unwieldy. From this one can see why Butler et al might have chosen to devise a grid-based solution to this in the form of *Refraction*'s tool's constraints editor (see Figure m in Chapter 4).

Figure kk: Excerpt from skill chain for The Casimir Effect, modelled in Microsoft Visio

9.6.2.4. What comes after a skill chain?

Aside from the difficulty of extracting useful information from the tree due to readability issues, there was the fact that a tree is far from a progression sequence. While the skill chain tells us the rules of progression, it is hard to imagine how one goes about extrapolating a sequence from a fairly complex hierarchical graph. Again, Butler et al provide a solution to this with their progression plan editor, essentially extending the skill chain concept. My experience with skill chains confirmed the value of Butler's additions. For my other case studies, therefore, I did not diagram skill chains but took them straight into progression planning.

9.6.3. From skill atoms to generic units

While I was prototyping micro-levels to progressively introduce gameplay I was also coming up with micro-levels that were interesting, but did not represent the introduction of new skills or concepts to the player and so were redundant to the immediate needs of my prototype's progression sequence or my skill chain. I stored these separately, knowing that I might need them later for other tools. Figure ll shows 5 of the 72 patterns I created and stored. Some were conceived of on paper and hand drawn; others were prototyped and screen captured. They are level design patterns: level design fragments that represent a particular configuration of one or more "skill atoms"/"types of gameplay" or what Butler would call "game concepts". In the centre of the top row is a pattern called "Timed door blocking trigger", for example. It contains several concepts, including door triggers, self-closing doors, and shooting a switch.

After a time, these design patterns, which began as the side effects of my process, began to drive it. When I looked at the micro-levels I had built to introduce skill atoms/game concepts, I came to see them merely as design patterns that introduce and demonstrate a given game concept. As new game concepts are necessarily experienced by the player in the context of old ones that they already know, I realised that deciding what an introductory pattern is was useless. This is because until my progression had been planned and designed, I would have had no way of knowing what old concepts the new concepts would be backgrounded by in the pattern.

Later, I moved the patterns into my progression design tool, where I was able to use my progression plan to give me the answers to exactly these questions. The patterns also came to form the basis of procedural content generation rules I created in *Ludoscope* – rules filtered

according to progression, using my progression design tool. In Figure gg one can see these workflow relationships. This is further discussed in Chapter 12.
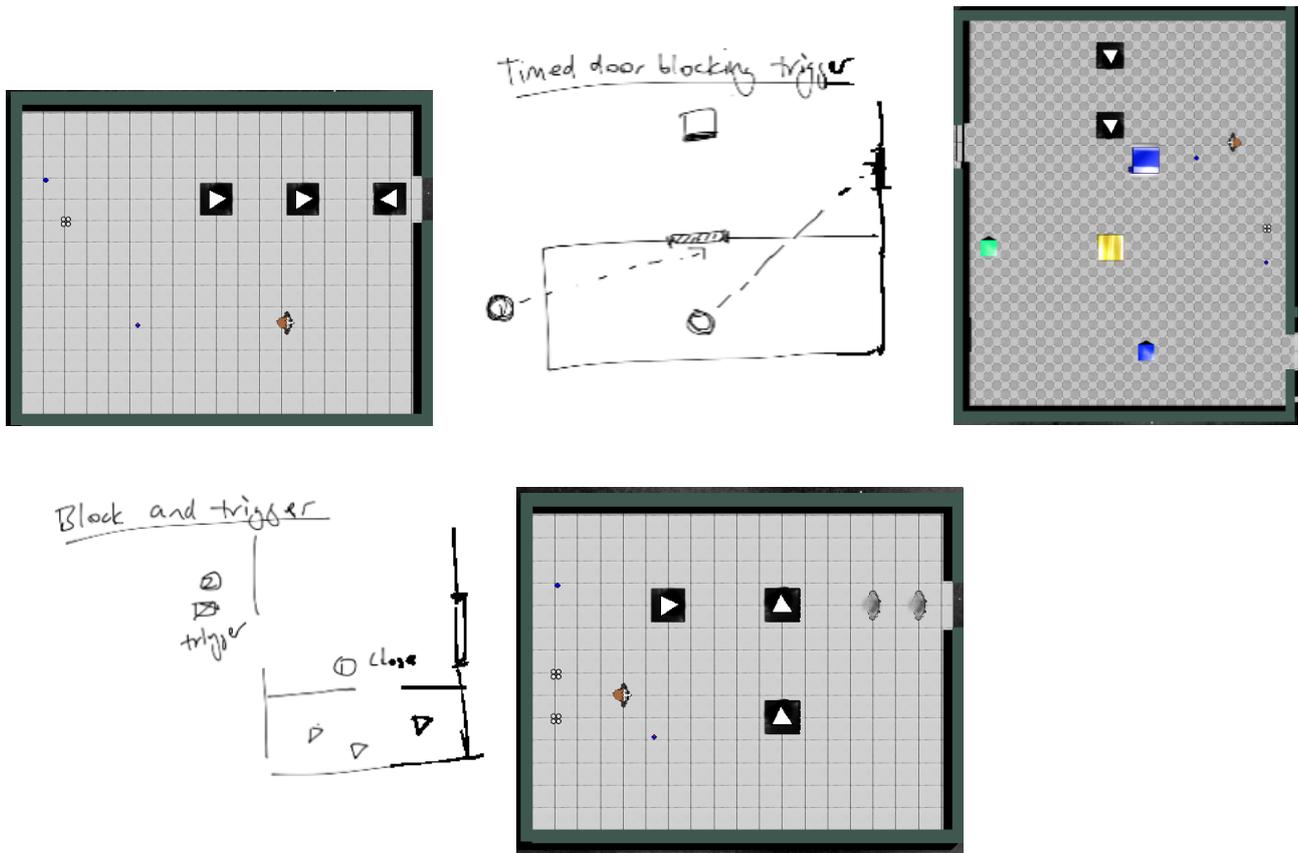

Figure ll: A selection from a collection of 74 level design patterns for *The Casimir Effect.*

9.7. Tools require preparatory work on design materials

Notice how, in my work on *The Casimir Effect*, I began to prepare my materials for use in tools (splitting my ideas into design patterns, for example) before I actually started using tools. I have observed that using tools affects design activities not solely during their use, but also before and after. These extraneous tasks of preparatory work pose challenges of their own, in addition to the challenges posed by actual use of the tool.

In my case – as is likely to be the case generally, as tools tend to formalise the design situation – preparatory design work was unpacking, restructuring/formalising and customising my design materials into a form suitable for use in the tools (at first, skill chains, but later progression planning and *Ludoscope*).

Tools require that implicit ideas be made explicit. They may require us to prepare design materials that exist in prototypes, on paper, or may yet only exist in the designer's head. They may be ideas that, in a conventional game development context, may only ever be expressed in shorthand in a document (their interpretation relying on implicit and shared understandings of a game genre), before being implemented by a programmer in the game.

9.8.   Connection between the design materials and the design product

The fact that the game design activity continues during production poses the question of where the design "lives" once production has started. Once a game is in production, pre-production documentation is now out-of-date, and it is likely that single-use documents are now being produced to flesh out details of specific parts of the design (as discussed in Chapter 2). In other words, there is likely to be no longer any single, coherent, up-to-date representation of the design situation. Therefore, as game production progresses, the game itself - though still incomplete - increasingly takes on the primary role of representing the design situation. Design activities and design thinking increasingly centre on the production materials as design materials.

This suits our current crafts-based design method. But the implicit goal of game design tools is to decouple design and production, allowing us to perform our design thinking using our own materials of design that exist separate to the built artefact.

I am not sure whether maintaining a parallel model of the design alongside production is useful, or potentially counter-productive (with the problems of maintenance, duplication and synchronisation, etc.) as it used to be during the era of large game design documents. One approach might be to consider materials generated in game design tools in the same way single-use game design documents are created – i.e. one-off representations of a part of the design situation, created for short-term use. Design materials for this purpose could be extracted from the in-production game to be manipulated for design thinking. Above I described something similar in extracting materials from prototyping to be understood in design tools, and in Chapter 9 I have described how I did this for game balancing in *Ultraworm*.

Another perspective is to assume that maintaining these design materials is useful, the benefits outweighing the costs. Arguably, the cost-benefit imbalance of maintaining documents is not entirely comparable: documentation, because its primary purpose (once written) is

communication, gives less value to a designer in return for the designer keeping it up-to-date. A "living" model or set of models of the design, by contrast, worked on every day, gives something back, retaining an ongoing use value. It is still useful partly due to the fact that, while the game may be now playable, it is still not primarily a design tool. For this reason, the models within design tools could be maintained as a test bed environment, for modelling and testing changes to the current design, before being signed off to go into the game.

I suspect that some aspects of the design situation – narrative and mission structures, for example – may be well served by maintaining their definitive, reference version of the design within tools. This is because these representations do not just serve for design thinking, but also design planning (*Refraction*'s progression planning being a good example of this). In this way, the design materials do not merely exist in parallel to production - they help drive and shape the project itself. Game design documentation conventionally performs this role to some extent, communicating implementation specifications to the rest of the team. But tools could potentially do a better job of this, driving the development process, providing a framework and a driver for design tasks performed during production, and maintaining a clear view of the "big picture" of the game. Again, we might recall how tools have this key advantage – a holistic view of the design situation - over an as-yet incomplete game. I have had some success taking this approach with *Articy:Draft* and with my own implementation of progression planning (discussed in Chapter 10), using my design representations to drive design subtasks and to some extent drive production.

9.8.1. Producing game assets with design tools

One way to maintain a connection between design materials for thinking and production is to use the same materials for both. In other words, export game assets from a design tool.

"Assets" is the name given to describe game content (art, audio, text, design data, and so forth) that is not built into the game executable, but is instead loaded as data files. Some of these data files will contain very large lists (and lists within lists) of value pairs that are used to initialise values used in the game (similar to a config or ini file). This can be numerical data: weapon reload times, jump heights etc. It can be data that defines relationships between entities (what inventory items a character might own, for example). Some of this design data might contain executable mission logic. The trend is towards putting more and more of the design of the game into data rather than

code – again, towards unravelling game designers' dependence on programmers and production concerns.

Arguably, the idea of maintaining synchronicity between the design materials (within a design tool, for example) and the product, as described above, means creating a kind of dependency between them. From this point of view, concretising this connection by having the design materials directly exported to be used as game assets would seem to be a natural thing to do.

One of the benefits of this is reducing the delay between making a design move and seeing the results in-game. As we know, some aspects of the design situation cannot be represented by design tools – game feel, for example – and for these, designers may want to playtest quickly to get feedback on those aspects.

Here I am thinking of design materials that concern data – the design data described above – rather than game systems. This data is usually produced with the help of tools anyway, but they are production tools. I am asking the question of whether these design game assets should be directly exported from design tools – i.e. the tools that designers would use to engage in design thinking.

There are a few issues with doing this, however. First, there is a workflow conflict to resolve: the designer would have to be assured that this link between the design and the assets did not make the designer afraid to change things for fear of breaking things in production.

Second, as we know, design materials and working with those design materials have certain particularities. Attempts to balance the exploratory and messy needs of design with the production needs of production, is a problem faced with 'accessible' game production tools. As discussed in Chapter (?), these needs can be irreconcilable. Ideally we would want to avoid turning design tools into production tools, intruding upon the design process by introducing potentially burdensome (quantity of labour) and intolerant production needs that have a negative effect on the design process. Some ideas for avoiding this are presented below.

Potentially, it could be a case of, as I have argued earlier in this chapter, needing to confront some of the received wisdom about game development and actively choosing to prioritise design thinking. In other words, we turn need to turn some of the current hierarchies on their head: unlike the production tools designers repurpose for design thinking, we frame our thinking about a

tool for a designer as *a design tool that can also be used to produce assets*, rather than *a production tool that also facilitates design exploration*.

9.8.1.1. Observation 11: Balancing design thinking with asset production

I noticed that I was able to somewhat take this approach in *Articy:Draft* for *The Particle Who Knew Too Much*. *Articy:Draft* is proposed as both a game design and a game production tool; the tool can be used for design thinking for narrative and mission structure, and it can also export these designs as data to be used in the game. I am exporting assets only for this, my adventure game project – for the other projects *Articy:Draft* remains purely a design thinking and planning tool. In this way, *Articy:Draft* serves me as a design tool, from which I am able to produce assets.

While my design materials are large and in a constant state of flux, I am able to export a small subset of them as a clean data set and get them into my game without having to reproduce them separately. To do this I had to make a number of decisions where I had the option of either prioritising the needs of design thinking or those of asset production.
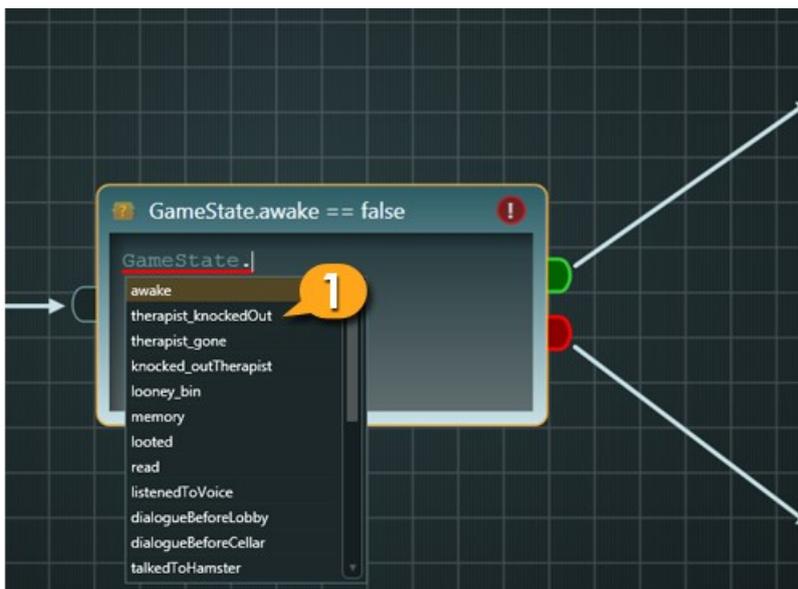

Figure mm: An example of *Articy:Draft*'s scripting function. Image taken from *Articy:Draft* documentation

Articy:Draft offers the designer the possibility of scripting game logic using their proprietary scripting language, and attaching these scripts to connections that have been authored within dataflow diagrams. This allows the designer to maintain, modify and check against the game state as it relates to game flow.

I was reluctant to use this feature for my game state logic. This feature would be fine for producing my game assets, but from the point of view of engaging with them as design materials for the purpose of design thinking, I felt that the cause and effect of state changes was not sufficiently clear in script form.

I wanted to find a way to use the graphical devices offered *Articy:Draft* to show them in a way that allowed me to visually parse my design more easily.

So when faced with the option of using scripting within *Articy:Draft*, which, from a production point of view seems like a good idea, I rejected it when I considered it from the point of view of its potential impact on my design thinking. Putting relationships into scripts would hide them from view, somewhat undermining the clarity of the graph-based visualisation. I instead spent time to devise an alternate, more readable graphical solution that I was able to use within *Articy:Draft*.

I did not want such a solution to limit my design outcomes, however. I approached the production of design data as if I were "coding" in a declarative, rather than an imperative (i.e. *Articy:Draft*'s scripting) language. Happily, I found that all my conditional and decision logic could in fact be expressed by simply creating GUI-based data fields to which I could assign visual elements, and in this way handle possible game state changes and conditions for my game events. I encountered a minor stumbling block: *Articy:Draft* does not offer the possibility of adding text-based or numerical properties to instances of data types. At first, this seemed as if it would be a serious problem, as I found myself forced to express quantitative elements, such as currency, as multiple instances of discrete units: e.g. a quest reward might be 5 units of 10 gold. While undoubtedly an inelegant and even counter-intuitive method as compared with adding the attribute "50" to a "currency" data type, it had the benefit of being highly readable. Much like coloured monopoly money set on the table before a player, it is quickly visually parsed as compared with a total noted on a pad of paper.

As a result of this highly graphical, "declarative" approach, my logic became highly visual and readable. It meant that I was able to quickly compare the form and content of quests against each other.

This data was interpreted by logic that I had authored exclusively on the game executable side, source code. The ease with which one is able to separate data and logic in this way is not

surprising, as creating content for most games of progression is about injecting different narrative content into repeatable modules of game logic and combinations thereof.

9.8.2.   Prioritising design thinking in dual-use tools

At the beginning of this chapter, I argued that the less visible and sometimes fragile needs of design thinking need to be championed within a context dominated by production needs. When making decisions that relate to both production and design it is tempting to prioritise practical, expedient solutions – i.e. production solutions. We should be conscious of this tendency and make proactive choices in our workflows and tools to prioritise design thinking.

This also applies to the decision-making of designers themselves. The decisions I needed to make in relation to producing assets in *Articy:Draft* demonstrate that the factors discussed above are relevant to a designer's choices too – not just choices made by the creator of the design tool. I have found that a designer must sometimes make conscious decisions to prioritise their design needs. In other words, prioritising one's own design needs requires some *reflecting-on-action*.

In *Articy:Draft* I had to be vigilant in order to impose this balance that preferences design needs, making conscious decisions to build materials in such a way as to facilitate the needs of design thinking, and prevent myself going down the seductive route of prioritising production.

Specifically, I made a series of decisions to prioritise the visibility of my design materials, keeping relationships between elements clear. While I could have used *Articy:Draft*'s scripting system for my purposes, I considered the cost to my design thinking, and managed to devise a way of avoiding it.

9.8.2.1. Scripts as design materials

Do we use scripting in cases where it is unnecessary? Had I not had the needs of the design process in my mind for this project, I would have probably scripted these state changes. I, like other developers, when designing a workflow, naturally gravitate to offering designers (in this case, myself) a "simple scripting language" for the authoring of game logic. It is, for production purposes at least, the simplest, most straightforward solution – i.e. for authoring and debugging. The back-and-forth and up-and-down-the-abstraction-hierarchy of a design conversation is not authoring and debugging, however.

The question of whether or not to use scripting is a good example of how design thinking needs are left out of the conversation. In the context of using simple game production tools for exploratory design thinking, the arguments I have heard about scripting concern the question of skill: where graphical interfaces are proposed for designers as an alternative to script-based interfaces, it is often on the strength of how to accommodate designers who lack programming skills. Aside from this need for the low viscosity of rapid, easy prototyping, design thinking needs are not explicitly considered. For a designer who happens also to be a programmer, scripting is often the most rapid, low viscosity option. From a production point of view, I personally far prefer scripting methods over graphical interfaces.

If these are materials I plan to use for design thinking, however, my preference changes. The far more important question from this perspective is whether, to the extent that a designer is using scripts as design materials, graphical representation may be more useful for a design conversation and facilitating the seeing-moving-seeing loop of design.

9.8.2.2. Observation 12: Accommodating messy data

The heart of the problem is that we need design materials to be different from production-ready assets. Game assets must be lean and efficient. Design materials, on the other hand, must be allowed to be in a state of flux and ambiguity in order to facilitate design thinking.

My adventure game design materials were no exception to this. When using *Articy:Draft* for this project one of my goals was to see if I could create this connection between the design materials and the game without compromising the fragile mess of my design thinking. This was largely achieved though treating the production assets as a subset of the design materials. This allowed me to pass quickly between my design materials and my game, seeing the results of design moves within minutes of making them.

This is not a new idea; Microsoft Excel is sometimes used by game designers in a similar way, where macros are written to export data from a subset of cells, leaving the designer free to write notes or do working calculations in others.

Filtering in this way within *Articy:Draft* would, ideally, be done by placing finished assets into a place in the tool to mark them as game ready – much as one would do using a folder on a desktop. *Articy:Draft* does not currently offer the possibility of tagging certain groupings for export; all data

is exported. I was therefore compelled to use the less efficient method of filtering the data at the point of import, with a view to optimising the data at a later stage.

9.9.   Game design tools do not have to do everything

If we step back to view design tools within the wider context of production we realise that game design tools do not have to do everything. Bearing in mind Hecker's exhortation for researchers to give developers "just two sticks and a rock" that can fit into an already complex tool chain, game design tools that perform a narrow, single task may well be useful.

In Chapter 10 I made a comparison between the design activity of evaluation and verification and validation style activities. I described in Chapter 9 how the task of game balancing, as a kind of verification of the design in its attention to detail and the type of questions it attempts to answer, can have different needs from a tool. Here I add to this argument from the pragmatic point of view of design and production workflow.

Evaluation on the one hand, and verification or game balancing on the other could be seen as being on a priority continuum that shifts from one to the other over the course of the project. By the time you are wanting to verify and validate, you are no longer revealing design problems that will require major design changes; you are revealing errors that one might see as "design bugs".

By the time we reach this latter stage, the project may well be well into production. This means we are likely to have the option of using the game. The upside of this is that the best simulation of the game can be the game itself. Automated testing is now commonly used to test design logic and data. Having built and used automated testing scripts I have seen how useful they can be for design - particularly for games-as-service projects that need to verify design data that is constantly being changed (i.e. similar to regression testing).

Game balancing is, generally, all about data. In a way, the difference between these two design activities or stages – i.e. design thinking and design testing/game balancing - mirrors the practical division we already make in implementation between hard-coded game system logic, and design data that is loaded as game assets. The hypothetical verification-type questions I gave above, for example, would usually be in data.

9.10. Single framework vs multiple tools

There are experiences I have had that seem to add weight to the idea of using a single framework. I have discussed how in game development we are already very aware of the dangers and the work involved in maintaining representations of the design situation and keeping them in sync with each other. There is also the added burden of learning more than one tool or language. Finally, I have raised the point that this requires from the designer skill and discretion to be able to make choices about the use of tools, and also to be prepared to explore in order to inform those choices (because they sometimes cannot be made in advance).

Other experiences I have had, however, make me tend towards another view. First, I have discussed how different tools give a different, filtered view of the design situation. I have also noted how tools attaining a comprehensive representation may be unfeasible. Crucially, a one-size-fits-all solution may be difficult due to the fact that, the form games take can be so varied, spanning a variety of rapidly evolving platforms and media. Finally, in this chapter I have discussed how design needs and tasks change over the course of a project.

Along the way I have also hinted at the logical conclusion of this: that, like designers in other disciplines, a game designer might ideally use a set of game design tools rather than a single tool. I am not alone in this thinking: Reyno and Cubel seem to suggest that a single form of representation may not be enough, saying "many theories and notations have unsuccessfully tried to capture the essence of gameplay in a single representation or diagram" (Reyno and Cubel 2009). I have to allow for the possibility, however, that this project – where I have used a toolset, rather than a single tool - has biased me towards favouring the toolset approach.

9.11. Tools to learn and grow with

As I have described in this chapter, working with procedural content generation – even thinking about working with it – fed into and informed my approach to progression and level design. It is inevitable that a tool, to some extent, imposes a process, and even a set of values, on a designer. Tools force one to think through a design "their" way. I have previously talked about this in terms of the negative effect of "tool bias". But there are positives too, in having a framework – however biased – imposed on a raw, informal style. I have used tools – but in some sense, tools have also used me.

This effect seems to linger, even after the work is done. Even if I never use these tools again, my experiences with them will continue to shape my approach to game design. I feel that I have learnt quite a lot about design from these tools, and been forced to think about my design ideas in new ways and from new angles.

Yet some of these tools take time and effort to master (*Machinations* and *Ludoscope* in particular), and I am not yet fully proficient with them. I do not see this as a problem, however, but rather as an opportunity. To expect game design tools to be easily mastered is, by association, based on an assumption that game design itself should be easy to master. It is an expectation that, ultimately, does game designers no favours.

Much earlier, I discussed how accessibility (a "low threshold") is seen as a virtue in some game production tools. I have discussed this in the context of the goal of empowering designers – accessibility is seen to enable designers to prototype and built their ideas without the need for programmers. This accessibility narrative expands beyond this though – i.e. beyond empowering game designers and towards empowering people who are not yet game designers but want to be. In her book *Rise of the Videogame Zinesters*, Anna Anthropy famously declared that the tools of game creation should be democratised because "everyone should be making games" (Anthropy 2012). *Twine* (Kilmas 2009), essentially a tool for creating interactive fiction at its core, is perhaps the ultimate in accessible game development tools, and has been particularly championed as a tool that provides an accessible, low barrier to entry for socially oppressed groups to access game development culture (Hudson 2014; Freeman 2015). Educators are now creating programs for children to learn to make games in schools (Koebler 2011). The idea is partly that by involving more people in the creation of games, we will produce better, more diverse and interesting games.

Accessibility is a good thing, but is only part of the equation. Great works of film, literature and visual art, for example, are not achieved because tools and techniques are easy to learn, and thousands of people can use them. They are the product, usually, of a minority of those people who have had the opportunity to not only learn, but master and deepen their craft – a process that takes many years. Tools should not just allow us to begin such a process towards mastery, they should help enable it.

10. *PROGRESSIMO*: A PROGRESSION DESIGN TOOL

10.1. Overview

In this chapter I explain how I built a progression design tool that builds upon Butler et al's "progression planning" tool concept and approach. My goal was to see whether the approach could be useful when applied to more demanding and more varied progression design problems than those posed by *Refraction*, a linear puzzle game. My method for this was to adapt and extend Butler et al's progression planning concept to support progression design for my game case studies.

For a functional overview of the tool see the Appendix. Below I describe my tool and how it builds upon, modifies and extends the model used by *Refraction*'s progression planning tool.

10.2. Goals

In Chapter 2 I discussed the increasing importance and complexity of game progression design.

The design of game progression – the player's journey through a game – is a core design task, applicable to nearly all contemporary videogame genres. It is a good candidate for design support, as the design of progression structures – which can be large scale and sophisticated – can present a complex, data-heavy task. Yet no computational support is available to designers.

In an attempt to fulfil this need for tools to support progression design thinking, Butler, Smith, Liu & Popovic have proposed a general architecture for "progression planning" tools which they have implemented within their level authoring tool for their educational game *Refraction* (Butler et al. 2013). As a demonstration of this concept, the *Refraction's* tool ably hints at its potential. Butler et al expect that progression planning tools could become more or less essential for very large, complex progressions where manual exploration becomes intractable. However, its strength as an approach to these challenging progression design problems remains largely untested. *Refraction* is a game that, given its genre and scope, arguably does not need the support of its progression planning tool to be designed. As Butler et al acknowledge, *Refraction* game's genre and scope as a puzzle game with modest progression design needs limits its applicability to other games. Butler et

al have suggested their progression planning approach could be used for other games, and I sought to test this notion.

I had two goals for this work. The first was to evaluate and explore how *Refraction*'s progression tool approach can be applied to the progression design challenges of games other than *Refraction* – particularly its universality in terms of servicing the needs of games of contrasting genres. The second, which foreshadows the first, is to discover and reveal the ways in which Butler et al's approach requires adapting or extending in order to apply it. This chapter focuses on the results of this second objective.

10.3. Method

It was not practical to use the *Refraction* progression planning tool itself as a starting point for my work, as it is integrated with the game's level editor and generation system. As I have explained, some of the tools I may have wanted to evaluate were unavailable or not practically applicable to my specific projects – hence I did not include them in this project. I considered that compared to the other tools in this category, however, Butler's progression planning is not difficult to implement. Moreover, by building my own version, I would have the ability to adapt it in order to make sure I could evaluate it from the point of view of all five case studies. I decided, therefore, to build my own progression planning tool based on *Refraction*'s approach.

Refraction's tool's progression planning was designed with a concrete game design problem in mind. I took a similar approach, in that I also designed my tool with concrete problems in view, my goal being to create a tool that could service the specific progression design needs of several of my game design case studies.

These needs were used to drive an iterative approach to designing and producing the tool: I began to design with the tool as soon as I had a working version of it running, and from then on made additions and modifications to it based on the needs of my case studies as they revealed themselves.

10.4. Progression design needs of the case studies

In Chapter 8 I described the progression design needs and challenges associated with each of my game design case studies. Usefully, these genres vary quite markedly in the nature of their

progression. Servicing these contrasting design needs helps enforce a degree of universality in my tool design. My small set of games cannot be representative of all game genres, however, and like *Refraction*'s system, the design of my tool was inevitably driven by the needs of specific games rather than the needs of all possible games. That said, my set of progression planning requirements go beyond the scope of those needed by *Refraction*.

For example, designing progression for *The Particle Who Knew Too Much*, requires:

- Managing the awarding of game inventory and currency, and using these variables to manage game progress and constrain design moves.
- Understanding how much or how little money a player would have at a certain point in the game, and setting the price of any purchases the player needs to make accordingly.
- Managing the order in which the player receives narrative knowledge within an open-world structure.
- Managing the gating and unlocking of new areas and missions.

Ultraworm has similar needs to those of *Refraction*: how to gradually introduce elements and combinations of elements in a linear fashion while managing difficulty.

The Casimir Effect requires a way to manage replayability. Within the game's branching spatial structure, I need the tool to show me where I need to add a temporary obstacle to gate an area in cases where the content in the area will be too difficult for the player at a given stage.

For *South Sea Trouble*, which has a semi-open mission structure I need to create a progression plan that allows the player to progress through the game without having completed every mission, while making sure that they have acquired all the skill level and knowledge they need to play missions that become unlocked.

In addition to my case studies, I was interested to see whether the tool could help with some of the design needs of freemium games built for the games-as-service model. Having designed for a freemium game using this model, I knew that a designer needs to, for example, ensure that

sufficient content is unlocked and available to the player at any given time, and be able to determine where the game's "pinch point"[29] might lie.

10.5. Approach

I have taken a "first, do no harm" approach that informed my design choices in adapting the tool.

As I have explained above, there exists a wide gulf between the primitive document-editing and non-standardised diagramming practices that game designers typically use and the advanced solutions proposed by the research community. While mixed-initiative game design research is exploring the potential of computers to participate in design thinking and serve not merely as "tool" but as collaborator or expert (Khaled, Nelson, and Barr 2013; G. Smith, Whitehead, and Mateas 2011b), practitioners still do their design thinking unsupported by tools. It is understandable that some of the solutions built are far ahead of current practice. In Chapter 5 I mentioned Smith's characterisation of the goals of this kind of research: the technology researchers like him have built is exaggerated in its futuristic and explorative nature for the purpose of demonstration; it is not built to be ready for immediate use in the field.

In keeping with my research goals my goal for this tool was quite different: rather than bold and exaggerated, I felt the need to make tool design decisions that rather erred on the side of small and tentative, imagined as the first steps a game designer might take with using a design tool for the first time.

One of the things that struck me about the progression planning in *Refraction*'s tool is its simplicity. With a *"first, do no harm"* philosophy in mind, it is the "low-tech" elements of *Refraction*'s progression planning tools - its simple constraints editing and visualisations – that seem powerful. Moreover, the concept of progression planning essentially formalises what many designers are more or less already doing, offering a domain-specific alternative to tasks that are typically performed using spread sheet software. As such it is a good candidate for this first small step in the direction of computer-aided design.

---

[29] A freemium game's pinch point is the point at which the player runs out of in-game currency and is compelled to decide whether to spend real currency to purchase more.

Indeed, game progression design does not need to be complicated. One way of thinking about progression design is as a difficult, yet unsophisticated data wrangling problem. In contrast with game progression, game systems may contain just a few simple rules, where the difficulty for a designer is in how to model and predict the emergent dynamics arising from those rules. Progression design, on the other hand, is usually more a matter of managing relationships between large amounts of data. In this context, the all-important "conversation with the materials of design" could be advanced even by simple tools that successfully help a designer enter, organize and visualize progression design data. And in some cases, this is what I have chosen to do: to simply visualize the data and allow the designer to conduct their own analysis and "policing" of design moves (as done with pen and paper) rather than add computational support.

Below I describe some of the features of *Progressimo*, and how they build upon Butler et al's *progression planning* concept. I focus here on key contributions. For a systematic description of the *Progressimo* and how it works, see the Appendix.

10.6. System model

As shown in Figure nn, the system built by Butler et al. comprises three component parts that allow the designer to edit their design from a high level of abstraction to concrete level designs: constraints, progression plan and the playable levels themselves. Butler et al. have used the term "constraints" to cover any type of design consideration – rules governing the concentration and ordering of game concepts, in the case of their implementation – that the designer wants to define and use to inform their progression plan.
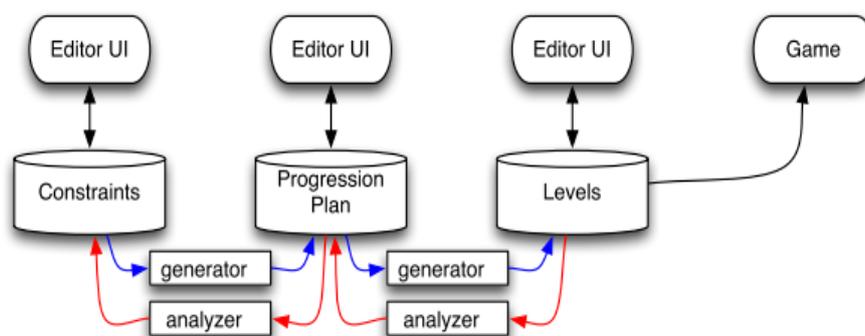

Figure nn: *Refraction*'s tool's system model (Butler et al. 2013)
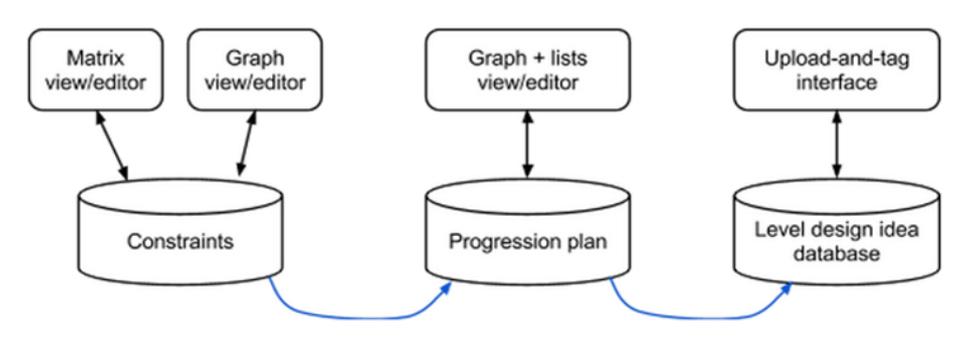
Figure oo: *Progressimo*'s system model

Progressimo's system, taking after *Refraction*'s tool, comprises constraints, plan and idea repository components (see Figure oo). Like *Refraction*'s tool, it invites the designer to define progression elements (Butler et al use the term "game concepts") and the relationships between them ("progression design rules", i.e. constraints) by editing them within a matrix/grid editor (see Figure pp), and uses this data to automatically derive and display further relationships. As the designer assigns progression elements to the plan for each "moment" (in *Refraction* these are game stages) in the progression plan component of the tool, the designer is guided based on the progression design rules. These lists of elements can then be used as a kind of "shopping list" when the designer goes to create the level.

10.7. Removal of level editing and procedural content generation

One of the most significant differences in my system, designed as it is to service a wide range of game genres, is that it does not include integrated level editor or procedural content generation system. *Refraction*'s system uses its constraints calculations to drive generative features and analyse design moves, generating the progression plan and, optionally, the levels themselves.

10.8. Integrated visualisation

Refraction's tool provides charts that allow the designer to see the rate at which new progression elements (game concepts) are introduced and the number of concepts used in each stage. I may add something similar in the future, as my analysis needs change towards the end of my projects (in my case a chart would be generated for each possible path though the progression structure). Up until now, however, this information did not feel like a priority for my progression design needs. These charts are particularly relevant to progression elements that focus on knowledge or

skill acquisition. While, for example, progression in *The Casimir Effect* is very much driven by skill acquisition, the time necessary to acquire the skills varies. Therefore, even if I introduced them in an even-paced way, from game level to game level, thus producing a smooth curve on a chart, it would not be a useful representation of the actual experience of skill acquisition unless I factor this data (i.e. time to master) in.

Due largely to the branching progression structures I found myself building, I chose to visualise the distribution and location of progression elements across the game within the progression plan (a graph) itself (see *Moment highlighting* in the Appendix).

10.9. Multiple views
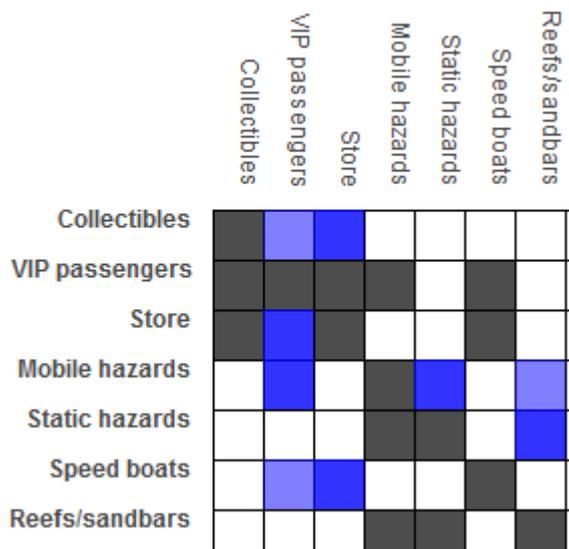
Figure pp: Part of the progression design rules editor in matrix view. Rules are for *South Sea Trouble*.
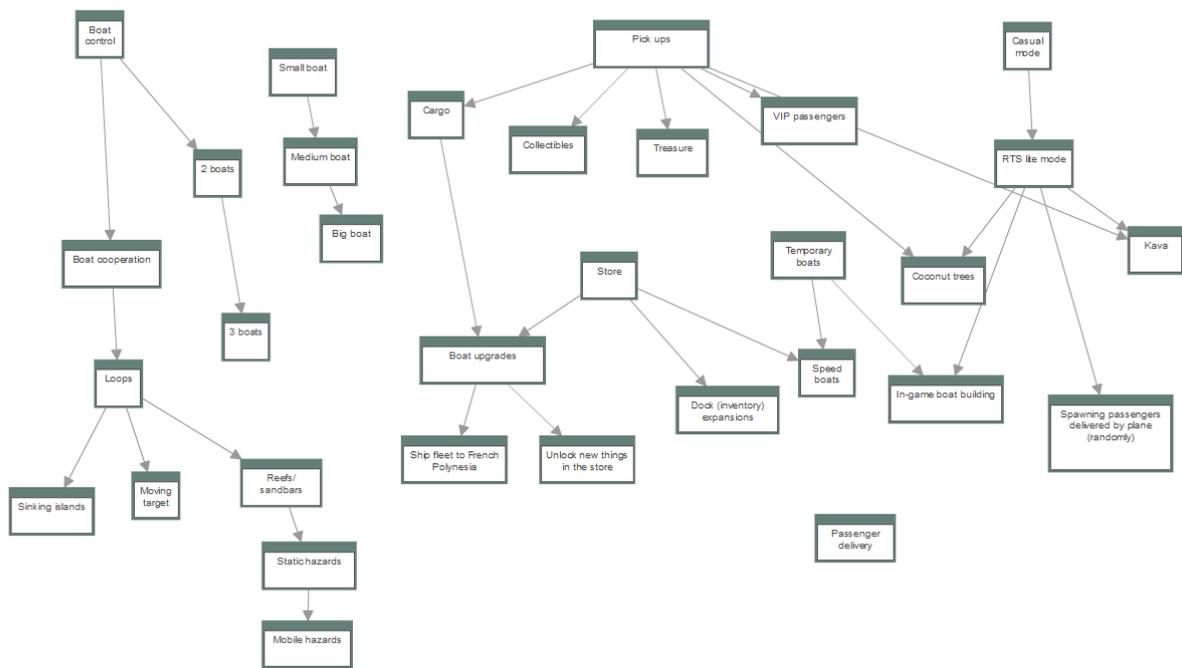
Figure qq: The progression design rules editor in graph view. Rules are for *South Sea Trouble*

Dan Cook's Skill Chain diagramming method, one of the inspirations for *Refraction*'s progression constraints component, is in graph form. While Butler et al.'s choice to present this in a matrix form instead provides a clean, easy way to edit and visualize relationships between game elements, I have chosen to also offer a graph-based view so that the designer has the option of visualising dependencies as chains for design that are useful to read or edit in graph form. As Hewett tells us, offering multiple alternative representations into the design situation can be valuable (Hewett 2005).

Figure qq shows the graph editor, which allows editing of prerequisite relationships in graph form. As in matrix view, graph edges are automatically added when constraints are transitively inferred by the tool. The designer can swap between both editors, which automatically update with any data changes.

10.10. Generic units

As outlined in Chapter 4, there have been efforts among game designers and within the research community towards the quantification of gameplay into units ("ludemes", "skill atoms", "verbs", etc.). The units bear similarities but they differ, filtered by the game design model they belong to. Accordingly, I have used the term "progression elements", to mean anything that the designer

chooses to use to structure their progression around. The designer can use multiple kinds of progression elements in the same project if they want to. Unlike in *Refraction*'s tool, these elements can be assigned to moments as quantities (numerical variables). (See *Progression elements* in the Appendix.)

Progressimo's unit of gameplay as defined within the progression plan is also generic: a "moment". In *The Particle Who Knew Too Much* a moment is a narrative event, for example, whereas a moment in Loot the Room is a dungeon level. (See **Moments** in the Appendix.)

10.11. Progression plan structure

The progression plan editor in *Refraction's* tool (see Figure m) accommodates a linear progression plan in the form of a table showing the elements contained in each stage of the plan's sequence. This suits a linear game like *Refraction*, but my case studies have non-linear progression structures. While *Refraction's* tool's progression plan editor serves as an important bridge between its constraint editor and its level editor, once constraints have been defined by the designer the progression plan for a game with a linear structure like *Refraction* could conceivably be created and analysed fairly easily by hand by the designer without computational support. By contrast, the ongoing task of ensuring that game elements present within a complex non-linear progression structure satisfy their progression constraints is intractable task for a designer to perform reliably on paper.

In order to accommodate the multi-pathed and open world structures of my games, I added a graphed-based progression structure editor and analyser. I use a graph traversal algorithm in order to apply constraints to non-linear level progressions. (See *Progression plan* in the Appendix.)

10.12. Progression state

State changes in *Refraction*'s tool comprised the addition of concepts to the player's accumulated knowledge. *Progressimo*, calculates a "progression state" for a moment. This state is calculated based that moment's and the preceding moments' progression element data (numerical variables, which can be negative). (See *Progression state information* in the Appendix.)

10.13. Gating gameplay

 "Gameplay gating"[30] is a concept familiar to game designers. Defining and managing gameplay gates (based on XP or currency, for example) is a common progression design task, and it fits naturally into the progression planning concept. *Progressimo* offers the ability to define gameplay gates. These are expressed in the form of progression elements.

10.14. Design gating

Refraction's progression planning uses progression design rules (constraints) to guide or constrain design moves. I have chosen to call these constraints "design gates", evoking the "gating" concept in game design. While "gameplay gating" is a well-known concept, my term "design gates" describes the effect of progression design rules on design moves. Gameplay gates are implemented in and imposed on the player by the game, during gameplay; design gates are defined only within the tool by the designer, to be imposed on the designer by the tool.

Having design gates and gameplay gates allows them to be considered together. It is useful to know whether or not the progression elements demanded by a gameplay gate are accessible to the player at the time at which they encounter the gate. Depending on the style of game the designer may use design gates to guide them on what they will need to impose on the player as a "hard" (i.e. a gameplay) gate.

10.15. Active evaluation

In Chapter 8 I discussed the function some tools provide of interactive simulation: the ability to interact with a simulation of the game system based on the model created in the tool. I suggested

---

[30] A "gameplay gate" restricts access to a part of the game until the player achieves something. For example, the game might "gate" a mission (leave it unlocked) or an area (physically gate it with an obstacle) until a certain progression state is achieved. E.g. the player needs XP Level 5 for the mission to unlock, or needs to have performed a certain task (e.g. killed a boss enemy) for the area to become available.

that instead of seeing this as a simulation of the player experience we could think of it as a kind of "active evaluation".

Here I have added a similar functionality – not to a simulation of a game system that reveal emergent dynamics – but to this model of game progression. This was instigated by a need to better understand the progression structure for *The Casimir Effect* – this was particularly important given its complicated unlocking structure. In "explore mode" the designer can simulate the completion of moments (game levels or missions) step by step. The designer "executes" the progression by clicking on accessible (unblocked) moments, step by step, in sequence. At each step where multiple moments are accessible, the designer makes a choice as to which moment to play. Access to moments are constrained by connections and gates. The graph is updated at each step with highlighting to show which moments are unlocked. Potential progression problems are also highlighted. (See *Explore mode* in the Appendix.)

Instead of error-checking, the designer is getting a general sense of whether their progression logic is working, not whether their design is free from errors.

I initially implemented an automated solution where this exploration was executed and problems discovered by the tool itself, in a single execution, using an exhaustive search algorithm. As previously discussed, I found the resulting experience for me, the designer, unsatisfactory. I found that with many possible paths through a structure, understanding and reading the results produced was time-consuming and difficult to read and understand. I felt out of control, that the tool's own "design thinking" and the explicit back talk it produced was too opaque and too fast for me to keep up with.

In fact, Smith notes the limitations and difficulties of raising the level of sophistication in automated analysis of models from the designer's point of view. Smith's *Ludocore*, is, of course, a very different proposition in terms of sophistication: its AI-driven automated analysis system factors in emergent dynamics of the game system itself into game progression, whereas I simply use estimates to stand in place of any game dynamics. (For example, for each level of *The Casimir Effect* I am estimating the amount of ammo used, based on playtesting. This is discussed in Chapter 12.) It does, however, cater for similar questions:

> The system's "gameplay trace inference" affordance allows a designer to ask for a symbolic gameplay trace, from the vast space of potential low-level action sequences possible in a game, that satisfies arbitrary logical constraints. In this representation, questions such as "is the game

winnable?", "how would someone get from here to there without using this special item", and even "how should I connect the various regions in my level design such that the player cannot escape without encountering all of my content?" all reduce to a common representation.

As Smith explains, however, this does require a lot of the designer.

> This capability, of course, is conditioned on the "designer" being an experienced logic programmer, the rules of the game being primarily symbolic as opposed to numerical, the requested properties of gameplay being expressible in a subset of first-order logic, and having to wait unreasonable lengths of time for the results of certain kinds of queries. It is on top of this admittedly shaky foundation that Ludocore makes its statements about AI in the game design process: deeply modeling videogames requires capturing not just the game's rules, but also the configuration of the game's world, a body of assumptions about the kinds of players who might play the game, yet another body of assumptions about the situation in play that currently interests the designer, and, on top of that, the ability to reason through all of this to generate concrete gameplay traces which tell the designer something that was out of range of their human inferential ability. In other words: deep computational modeling of gameplay is hard, but possible.

Smith, like other game design tool researchers, created *Ludocore* to show what is possible. *Progressimo*'s explore mode – very primitive by comparison – sacrifices depth and sophistication for direct designer control as a form of "active evaluation". It is an interactive mode that allows the designer to explore potential implications and some of the very basic emergent properties of their design at their own pace. Taken in the context of my very humble goal – i.e. to provide a way of understanding design outcomes that is superior to calculating them in a spreadsheet – I am pleased with the outcome.

As to depth and sophistication, I take the pragmatic point of view that I expressed in the previous chapter: in many cases, some of the more precise, specific design questions that would require automated analysis might be best left to automated testing, using the game code itself to provide answers based on its own simulations. My feeling is that if my design question or test scenario is too complex and unseeable to answer via my explore mode it may be more of a verification and validation type of question (e.g. "are there any places where the player will get stuck", "is it possible to complete the game with this specific combination of skills and inventory") to be resolved later as game balancing and design refinement or bug-resolution tasks. In other words, they are likely to be the kinds of design questions that I would ordinarily want to check at a later stage in the design and development process, by which time automated testing could provide a more accurate answer.

Another advantage of an explorative, "hands-on" approach to querying the design materials is that the designer can define for themselves the nature of the issues they are hoping to reveal — and

this set of requirements can change fluidly and intuitively in the designer's own mind, rather than input as questions to the tool. The designer is effectively devising and carrying out their own "test cases". A single exploration constitutes a test case – the parameters of which are chosen by the designer directly by choosing moments to "execute". The designer can even revise and reformulate their test case or question while they are asking it.

This makes a difference to the viscosity of design thinking. Say, for example, we have a question such as "what is the number of missions the player will typically have available at any given time, using a 'completionist'[31] play style" and imagine asking it of my abandoned exhaustive search-based method: in the time taken for the designer to think through and input their query and run the simulation, and read the results, they might have intuitively (i.e. not even, perhaps, fully consciously) already have asked and answered this question themselves via "active evaluation".

10.16. Storage and retrieval powered by progression design rules

I have described how for *The Casimir Effect* I built up a collection of level design patterns, and initially stored these in a notebook using *Evernote*. Once I had begun to design the game's progression, working out which progression elements (in the case of *The Casimir Effect*: game verbs, puzzle types, entity types) would be included in each level, it became obvious that sifting through my notebook looking for patterns to use when building my levels would be difficult and time-consuming.

The Notebook component (the idea repository component in Figure oo) provides a way of storing such ideas in a way where they are easy to find again at the right moment, thanks to a tagging and filtering system. The notes (which are images) can be tagged with the progression elements that they contain, enabling the tool to filter these down to display a subset of notes containing only the progression elements contained in the selected level. In this way, the notebook component also addresses another universal design phenomenon - the intimidating "blank page" - by providing existing design materials as a starting point.

---

[31] A "completionist" is a player who tends to aim for 100% completion of all game content (e.g. collect all items, unlock all skills, etc.).

While a progression planning tool can suggest and help structure a logical and orderly design workflow, design researchers have shown that it is, in reality, impossible to wedge the complex reality of real-life design process into a linear workflow. Designers find themselves generating the right ideas at the wrong moments (which could be called "side effects", discussed in Chapter 3); while working on a given level of the game, designers can often find themselves generating or discovering design solutions that, while inappropriate for that level, might be a good candidate for inclusion in another part of the game at some point in the future. These might be level design fragments, work-in-progress levels that do not yet have a home within the progression plan itself.

The conventional way for a designer to deal with inopportune but good ideas is to note down them down before returning to the task at hand. Creativity support tool researchers propose that design tools improve upon this, and "provide the creator with a kind of virtual notepad" to record and store ideas "as they come to mind"(Lubart 2005). I have attempted to do this with *Progressimo*'s Notebook component. The designer should feel unrestricted to generate ideas that do not quite "fit" yet, without an unconscious feeling instilled by the formalisation of the representation that this work is "off-task", or likely to be lost or wasted. As well as tolerating the all-important unfinished-ness and ambiguity in the design situation by accommodating, the Notebook feature serves to legitimise the anarchy of the process in that it grants the designer built-in "permission" to work outside the bounds of the formal constraints of the progression plan.

10.17. Integration within the design workflow

For *The Casimir Effect*, I am using progression data created in *Progressimo* to inform a rule-set for procedural content generation in another tool: *Ludoscope*. To generate a game level, I use the relevant moment's list of both explicitly defined and inherited progression elements to filter a master list of rules I have devised to create game levels (this relationship can be seen in the workflow diagram in Figure gg). This is similar to how *Refraction*'s progression planning component integrates with its level generation system, except instead of having two components within the same tool, the data is passed between my progression planning tool and *Ludoscope* via an exporter I wrote for this purpose.

As well as using data produced by the tool, I am feeding data into it. I have been using information about game progression that I have derived from playtesting to estimate changes to the progression state for a given level. This is similar to the way I have used, on occasion, estimates

derived from playtesting in my *Machinations* models. Here again, data generated from playtests can stand in for (i.e. simulate) dynamics that are not being generated by the model, in the case where the model is incomplete. For example, for *The Casimir Effect* I need to estimate the minimum and maximum amount of surplus ammunition the player can accumulate over the course of the game, and to help with this I have been feeding averages of playtesting results (i.e. how much ammo the player typically has left at the end of a given level) into my progression plan as data for these kinds of calculations.

A more detailed and systematic breakdown of how *Progressimo* functions can be found in the Appendix.

10.18. Next steps

So far, by building *Progressimo* around the needs of my case studies, I have found it to be useful. In other words, I have found the core idea of progression planning together with my adaption and extensions, has offered meaningful support for the progression design of varied genres as represented by my five case studies. There are still some features I would like to add to explore the possibilities for interoperability – specifically the importing of playtesting data and the exporting of progression-related design data for use as game assets. Beyond this, a logical next step in this work is to see if a generic approach could be developed further to support a wider range of designers and their designs. It would make sense for a similarly iterative development process to be used for this, involving designers as beta testers to inform further development.

11. CONCLUSION

The adoption of game design tools, I have argued, would represent a significant evolution in current practice, from a crafts-based practice to a self-conscious design. With this project I add to the field of work on game design tools by contributing evaluation research from a practitioner perspective. This work was framed by the following research questions:

1. Can game design tools improve game design practice?
2. Can game design tools expand the scope of game design outcomes?
3. How do design tools change or impact the game design process?

More generally, I hoped to discover knowledge that could contribute to future game design tool development research.

To answer these questions, I took the point of view of a practitioner, undertaking practice-led evaluation research. Using a method of participant-observer/"reflective practitioner", I applied game design tools to longitudinal game design case studies. Having recorded and reflected upon my design work with a range of game design tools, I drew from this material twelve selected observations that touched on various aspects of my experience with them. I analysed and framed these observations with a discussion of evaluation themes I developed drawing on game design research literature, Design Studies and Creativity Support Tool research.

11.1. Summary and significance of observations

Based on my experiences, I came to a number of conclusions about game design practice with tools.

Using game design tools can be challenging. I developed the view the game designers need to develop better formal understanding of game systems in order to understand their representations in tools. It is useful for tools themselves to offer paths or bridges to formal literacy within the tool. The simulation of gameplay dynamics, for example, seems to offer a good entry point. Tools that build upon semi-formal existing design practices – such as flow-charting – also provide a way in.

By challenging a designer, and providing the designer a way of meeting the new challenges tools present, tools can have a positive effect on practice. From notions of game design – subjective as they may be – codified into these tools, I felt that I gained new knowledge about potential game design processes and game design process in general. Being compelled to externalise and formalise my thinking impacted my design thinking not only when I was using tools, but this influence extended to my design thinking and activities even when not using tools, permanently changing the way I approach design. This makes me think that tools offer the possibility of framing a journey towards deepening knowledge over the long term.

In addition, using tools seemed to afford me new design freedoms and confidence. I found that by defining my own design rules I could make design moves more securely, and experience greater freedom to explore owing to the fact that I could leave it to a tool to remind me of my boundaries. Procedural content generation in particular afforded me freedom to work at different levels of abstraction concurrently.

Game production organises design materials; having game design support for organising and structuring my design materials, however, meant that I could safely create more design materials without needing the context of production. Perhaps one of the most powerful aids to me has been simple visualisation – visualisations of the design situation are a major step up from current practice. Tools help provide structure for process as well: I discovered how a tool can provide a workflow to help concepts along a path towards final concrete gameplay.

Within this context of added structure and security, computational support had the effect of expanding the scope of game design outcomes, allowing for more complexity in my progression logic and game systems designs.

This security and confidence afforded is not total, however. I discovered that I needed to exercise critical judgement in relation to the representation of my designs as viewed through the filter of a tool, as I found that a tool does not give a designer a full or even necessarily a true account of the "fun" of their design. A designer must develop literacy not just to use tools but also to use them wisely.

Indeed, these tools require that the designer take a reflective point of view not only in relation to their design activities but also their use of the tools: a designer should be mindful so as not to let

themselves become distracted or drawn into "busy" work that feeds the tool rather than the needs of the design.

So while tools may offer benefits, they can require additional effort from the designer. This involves not only effort required by the tools in terms of learning and usage, but also preparatory work on the design materials prior to use, i.e. so they can be represented and manipulated in the tools.

I have addressed some aspects of how design tools might be integrated into existing design and development workflows, noting that game design tools do not have to do everything for a designer, as a designer has a range of production-reliant tools – including prototyping, player metrics, automated testing – that may be more suitable for performing certain kinds of design tasks. I have argued that, similarly, no single design tool need cater for all design thinking needs. A toolset, rather than a single design tool, may be the ideal.

I found that tools can replace some, but not all, roles that prototyping currently serve in game design practice. In addition, prototyping and design tools can complement each other: ideas generated from prototyping can be channelled into tools and playtesting data can be input into tools to inform and complete design models.

It is helpful for a tool to enable the designer to participate in design thinking processes at "human speed" – I found that when a tool is "cleverer" than the designer it can, in some contexts, be counter-productive. I have proposed that we focus on evaluation and analysis when supporting design thinking, and perhaps leave validation and verification type tasks to other kinds of tools.

By building my own tool (*Progressimo*) I found that it is possible to apply and adapt Butler's progression planning approach to other game genres, by using my case studies and their concrete progression planning challenges to discipline and inform this process. I developed some ideas for extending the concept, which I incorporated into my tool.

As a solo researcher, there have necessarily been limitations on what I have been able to achieve in this project. Time spent on game production tasks came at the expense of game design activity, and meant that I was not able to make as much progress with my case studies as I would have liked. More progress would have given me the opportunity to test tasks that arise more towards the end of the development cycle such as game balancing. Ideally, I would have undertaken this

research as a designer with a production team. This would have lessened the load in terms of non-design development (i.e. labour that had to be performed but did not contribute to research) and would have also been more representative of a team experience.

11.2. Future research directions

I was not able to represent all game design tools and approaches in this study. This was in part due to the limited time and scope of the project, but also due to the nature of game design tool research itself: often a software tool is not intended or designed for use outside of the specific research validation scenario for which it is built. One of the tools – *The Sentient Sketchbook*, an extremely useful looking tool that I hope to use in some form in a future project – I attempted to use but it became clear that there was a mismatch between specific features of the tool and the feature of the case study I had intended to use it for. While this is completely understandable from the point of view of a researcher's immediate research goals, it would be beneficial if we could as a field encourage and find ways to enable researchers to facilitate third party practice-led evaluation of their work. At the very least, this could enable researchers developing new work in this area to better review previous work.

There is still much scope for dedicated practice-led evaluation research in this area. Even at a personal level, I am yet to master these tools, and they have only just begun to change the way I design. In view of this, it is my intention to spend more time with them exploring their possibilities and observing how my design practice develops.

Having found Butler's progression planning concept as well as my own adaptions and additions useful for my case studies, an obvious next step is to engage in practice-led evaluation involving other designers and their games, and perhaps further adapting the tool to accommodate some of the range of progression design problems that they encounter.

In the course of this project I have become interested by the idea of exploring potential relationships between production-dependent design tools and methods – e.g. playtesting prototypes and player metrics – and design thinking tools. There is a range of design data currently being used within the context of analytics tools, in the service of data-driven design. A question for future research is whether such data might be used in the context of game design tools, in the

service of self-conscious game design practice? In short, what if, via design thinking tools, data science could be brought somehow more under the control of designers?

Right now many game designers are suspicious and even feel threatened by the use of technology in design, as it is sometimes used against them, or even to replace them. The problem of how a tool-supported design practice can be adopted by a significant section of the game design community (and from there, give rise to a practitioner discourse) remains a significant challenge and one worthy of further enquiry.

BIBLIOGRAPHY

Adams, Ernest, and Joris Dormans. 2012. *Game Mechanics: Advanced Game Design*.
http://books.google.com/books?hl=en&lr=&id=_Azio0txIdAC&oi=fnd&pg=PR7&dq=Game+M
echanics:+Advanced+Game+Design&ots=RmlGUK8-YI&sig=4AKaB1nSZyLH0xzkrHxgKklcu6o.

Agustin, Michael, Gina Chuang, Albith Delgado, Anthony Ortega, Josh Seaver, and John W.
Buchanan. 2007. "Game Sketching." *Proceedings of the 2nd International Conference on
Digital Interactive Media in Entertainment and Arts - DIMEA '07*. New York, New York, USA:
ACM Press, 36. doi:10.1145/1306813.1306829.
http://portal.acm.org/citation.cfm?doid=1306813.1306829.

Alexander, Christopher. 1964. *Notes on the Synthesis of Form*. Harvard University Press.
http://books.google.com/books?id=Kh3T3XFUfPQC.

Alves, Valter. 2013. "Design Patterns in Games : The Case for Sound Design." In *Second Workshop
on Design Patterns in Games*.

Amant, Robert St., and Thomas E. Horton. 2002. "Characterizing Tool Use in an Interactive
Drawing Environment." *… of the 2nd International Symposium on Smart …*. ACM, 86–93.
doi:10.1145/569005.569018.
http://portal.acm.org/citation.cfm?id=569018\nhttp://dl.acm.org/citation.cfm?id=569018.

Anonymous. 2015. "The Scratchware Manifesto." Accessed May 28.
http://www.homeoftheunderdogs.net/scratch.php.

Anthropy, Anna. 2012. *Rise of the Videogame Zinesters*. Seven Stories Press.

Aponte, M.V., Guillaume Levieux, and Stephane Natkin. 2011. "Measuring the Level of Difficulty in
Single Player Video Games." *Entertainment Computing*. Elsevier.
http://www.sciencedirect.com/science/article/pii/S1875952111000231.

Araújo, Manuel, and Roque. 2009. "Modeling Games with Petri Nets." In *Breaking New Ground
Innovation in Games Play Practice and Theory Proceedings of the 2009 Digital Games*

226

Research Association Conference, edited by Barry Atkins, Helen Kennedy, and Tanya Krzywinska. Brunel University. http://www.digra.org/dl/display_html?chid=09287.37256.pdf.

Archer, L. B. 1965. *Systematic Method for Designers*. Council of Industrial Design.

Asimov, Morris. 1965. *Introduction to Design*.

Beck, Kent, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, et al. 2001. "Manifesto for Agile Software Development." *The Agile Alliance*. URL: http://agilemanifesto.org/. http://agilemanifesto.org/.

Benington, Herbert D. 1956. "Production of Large Computer Programs." In *ONR Symp. Advanced Program Methods for Digital Computers*, 15–27. doi:10.1109/MAHC.1983.10102.

Bertini, Enrico, and Catherine Plaisant. 2006. "Report on BELIV ' 06 Workshop Beyond Time and Errors : Novel evaLuation Methods for Information Visualization." In , 1–5.

Birdwell, Ken. 2006. "The Cabal: Valve's Design Process for Creating Half-Life." In *The Game Design Reader: A Rules of Play Anthology*.

Björk, Staffan, and Jussi Holopainen. 2005. *Patterns in Game Design*. 1st ed. Hingham Mass.: Cengage Learning.

Blinn, Scott, Stephane Bura, Nicholas Fortugno, Scott Brodie, Daniel Cook, and Victor Jimenez. 2008. "Group Report: Multiplayer Game Atoms." *The Third Annual Game Design Think Tank Project Horseshoe 2008*. The Avallone Media Group. http://www.projecthorseshoe.com/ph08/ph08r5.htm.

Bojin, Nis. 2010. "Ludemes and the Linguistic Turn." *Language*, 25–32.

Bura, Stephane. 2006. "Game Diagrams." http://users.skynet.be/bura/diagrams/.

Butler, Eric, Adam M. Smith, Yun-en Liu, and Z Popovic. 2013. "A Mixed-Initiative Tool for Designing Level Progressions in Games." In *ACM Symposium on User Interface Software and Technology*. http://dl.acm.org/citation.cfm?id=2502011.

227

Chiang, Y. C., and M. H. Wang. 2005. "The Capture of Design Eureka." In *International Workshop on Studying Designers '05*.

Church, D. 1999. "Formal Abstract Design Tools." *Game Developer* 3: 28. http://www1.cs.columbia.edu/~cs4995/files/Doug_Church_FADT.pdf.

Cook, Dan. 2006. "Lost Garden: Persistent Myths about Game Design." http://www.lostgarden.com/2006/10/persistent-myths-about-game-design.html.

———. 2007. "Gamasutra - Features - The Chemistry Of Game Design." *Gamasutra*. http://www.gamasutra.com/view/feature/1524/the_chemistry_of_game_design.php.

Costikyan, Greg. 2002. "I Have No Words but I Must Design: Toward a Critical Vocabulary for Games." *Computer Games and Digital Cultures Conference*, 9–33. http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:I+Have+No+Words+&+I+M ust+Design+:+Toward+a+Critical+Vocabulary+for+Games#0\nhttp://andrey.savelyev.2009.ho mepage.auditory.ru/2006/Ivan.Ignatyev/DiGRA/I Have No Words %26 I Must Design_Toward a.

———. 2005. "Death to the Games Industry: Long Live Games." *The Escapist*. http://www.escapistmagazine.com/articles/view/video-games/issues/issue_8/50-Death-to-the-Games-Industry-Part-I.

Cousins, Ben. 2004a. "SELFCONSCIOUS AND UNSELFCONSCIOUS CULTURES." *Weapons of Mass Disruption*. https://benjaminjcousins.wordpress.com/2014/04/01/selfconscious-and-unselfconscious-cultures/.

———. 2004b. "Elementary Game Design." *Develop Magazine*, October, 51–54. http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Elementary+Game+Design #4.

Crawford, Chris. 2002. *The Art of Interactive Design*. *A Euphonious and Illuminine Building Successful Software*. Vol. 25. No Starch Press, Inc.

———. 2003. "Chris Crawford on Game Design." *Computer*. doi:10.1093/carcin/bgs054. http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-

228

20&path=ASIN/0881341177\nhttp://books.google.nl/books?id=_0BQFquvAC8C\nhttp://www.vancouver.wsu.edu/fac/peabody/game-book/Coverpage.html.

Cross, Nigel. 1997. "Creativity in Design: Analizing and Modeling the Creative Leap." *Leonardo* 30 (4): 311–17. doi:10.2307/1576478.

———. 2006. *Designerly Ways of Knowing*. *Designerly Ways of Knowing*. Springer London.

———. 2008. *Engineering Design Methods: Strategies for Product Design*. *Design*. John Wiley & Sons Inc. http://www.amazon.de/dp/0470519266.

De Peuter, Grieg. 2014. "Beyond the Model Worker: Surveying a Creative Precariat." *Culture Unbound: Journal of Current Cultural Research* 6: 263–84. doi:10.3384/cu.2000.1525.146263. http://www.cultureunbound.ep.liu.se/v6/a15/.

Do, Ellen Yi-Luen, and Mark D Gross. 1996. "Drawing as a Means to Design Reasoning." *Artificial Intelligence in Design '96 Workshop on Visual Representation, Reasoning and Interaction in Design, Palo Alto, CA.*, no. June: 1–11. papers2://publication/uuid/02F84179-3333-494B-8588-D4621BCB4CAD.

Dormans, Joris. 2009. "Machinations: Elemental Feedback Structures for Game Design." In *Proceedings of the GAMEON-NA Conference*, 20:33–40. http://journals.hil.unb.ca/index.php/GC/article/viewArticle/3787.

———. 2010. "Adventures in Level Design: Generating Missions and Spaces for Action Adventure Games." In *Foundations of Digital Games Conference 2010*. http://www.jorisdormans.nl/pdf/dormans2010_AdventuresInLevelDesign.pdf.

———. 2011. "Simulating Mechanics to Study Emergence in Games." *Artificial Intelligence*, 2–7.

———. 2012. "Engineering Emergence: Applied Theory for Game Design." *Engineering of Complex*. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1690358.

Dormans, Joris, and Sander Bakkes. 2011. "Generating Missions and Spaces for Adaptable Play Experiences" 3 (3): 216–28.

Dormans, Joris, and Stefan Leijnen. 2013. "Combinatorial and Exploratory Creativity in Procedural Content Generation." *Paper Presented at the PCG Workshop at the FDG Conference*. http://www.jorisdormans.nl/article.php?ref=creativity.

Dorst, Kees. 2003. "The Problem of Design Problems." *Design* 4 (Cross): 135–47. doi:10.1504/JDR.2004.009841. http://research.it.uts.edu.au/creative/design/papers/23DorstDTRS6.pdf.

Dorst, Kees, and Nigel Cross. 2001. "Creativity in the Design Process: Co-Evolution of Problem-Solution." *Design Studies* 22 (5): 425–37.

Dubberly, Hugh. 2004. *How Do You Design? A Compendium of Models.*

Evernote Corporation. 2008. "Evernote."

Fahey, Rob. 2013. "In Defence of Data-Driven Design." *Gamesindustry.biz*. http://www.gamesindustry.biz/articles/2013-11-15-in-defence-of-data-driven-design.

Falstein, Noah, and Hal Barwood. 2006. "The 400 Project." http://www.theinspiracy.com/400_project.htm.

Fischer, Gerhard. 2004. "Reflective Practitioners and Unselfconscious Cultures of Design Department of Computer Science , 430 UCB," 1–4.

Flurry. 2005. "Flurry Analytics."

Freeman, Will. 2015. "Twine Lets You Pen Your Own Adventure and Share It across the Net | Technology | The Guardian." *The Observer*, July 10. http://www.theguardian.com/technology/2015/jul/10/twine-interactive-stories-games.

Fullerton, Tracy. 2008. *Game Design Workshop: A Playcentric Approach to Creating Innovative Games*. Elsevier Morgan Kaufmann.

"GAME SEEDS." 2015. *2010*. Accessed June 27. http://www.gameseeds.net/.

GameAnalytics. 2010. "GameAnalytics." http://www.gameanalytics.com/.

Goldschmidt, Gabriela. 1994. "On Visual Design Thinking: The Vis Kids of Architecture." *Design Studies*, no. 1966: 158–74. http://www.sciencedirect.com/science/article/pii/0142694X94900221.

Greenberg, Saul, and Bill Buxton. 2008. "Usability Evaluation Considered Harmful (some of the Time)." *Proceeding of the Twentysixth Annual CHI Conference on Human Factors in Computing Systems CHI 08*, 111. doi:10.1145/1357054.1357074. http://portal.acm.org/citation.cfm?doid=1357054.1357074.

Grünvogel, Stefan M. 2005. "Formal Models and Game Design." *Game Studies* 5 (1). Citeseer: 1–9. http://www.gamestudies.org/0501/gruenvogel/.

Hagen, Ulf. 2011. "Designing for Player Experience: How Professional Game Developers Communicate Design Visions." *Journal of Gaming & Virtual Worlds*. http://www.ingentaconnect.com/content/intellect/jgvw/2011/00000003/00000003/art00006.

Hecker, Chris. 2011. "A Game Developer's Wish List for Researchers - Chris Hecker's Website." http://chrishecker.com/A_Game_Developer's_Wish_List_for_Researchers.

Hecker, Chris, Damián Isla, Borut Pfeifer, Steve Rabin, and Stuart Reynolds. 2009. "The Photoshop of AI: Debating the Structure vs Style Decomposition of Game AI." In *Game Developers' Conference AI Summit*. http://www.gdcvault.com/play/1249/(307)-The-Photoshop-of-AI.

Hewett, Tom T. 2005. "Informing the Design of Computer-Based Environments to Support Creativity." *International Journal of Human-Computer Studies* 63: 383–409. doi:10.1016/j.ijhcs.2005.04.004. http://www.sciencedirect.com/science/article/pii/S1071581905000431.

Hewett, Tom T., Mary Czerwinski, Michael Terry, Jay Nunamaker, Linda Candy, and Bill Kules. 2005. "Creativity Support Tool Evaluation Methods and Metrics." *Creativity* 72. Duke University Press: 10–24. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.91.8123.

Holmgard, Christoffer, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. 2014. "Personas versus Clones for Player Decision Modeling." In *13th International Conference, ICEC 2014*.

Holopainen, Jussi, T Nummenmaa, and Jussi Kuittinen. 2010. "Modelling Experimental Game Design." *Nordic DIGRA 2010*. http://www.digra.org/dl/db/10343.48507.pdf.

Hudson, Laura. 2014. "Twine, the Video-Game Technology for All - The New York Times." *The New York Times Magazine*, November. http://www.nytimes.com/2014/11/23/magazine/twine-the-video-game-technology-for-all.html?_r=0.

Huizinga, Johann. 1950. *Homo Ludens: A Study of the Play - Element in Culture*. *A Study of the Element of Play in Culture*. Routledge. doi:10.1177/0907568202009004005.

Hunicke, Robin, Marc Leblanc, and Robert Zubek. 2004. "MDA: A Formal Approach to Game Design and Game Research." *Discovery* 83 (3). AAAI Press: 04–04. doi:10.1.1.79.4561. http://www.aaai.org/Papers/Workshops/2004/WS-04-04/WS04-04-001.pdf.

IGDA. 2004. "Quality of Life in the Game Industry: Challenges and Best Practices." *April*. http://legacy.igda.org/sites/default/files/IGDA_QualityOfLife_WhitePaper.pdf.

Järvinen, Aki. 2005. "Theory as Game: Designing the Game Game." *Changing Views Worlds in Play Proceedings of the 2005 Digital Games Research Association Conference*, 10. http://www.digra.org/dl/db/06276.43287.pdf.

———. 2008. "Games Without Frontiers: Theories and Methods for Game Studies and Design." *Presented as PhD Thesis in University of Tampere,* 7. http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Games+without+Frontiers:+Theories+and+Methods+for+Game+Studies+and+Design#0.

Juul, Jesper. 2002. "The Open and the Closed: Games of Emergence and Games of Progression." In *Computer Games and Digital Cultures Conference Proceedings*, edited by Frans Mäyrä, 323–29. Tampere University Press. http://www.digra.org/dl/db/05164.10096.pdf\nhttp://www.jesperjuul.net/text/openandtheclosed.html.

Kadoudi, Omar. 2014. "The Creative Process of an Oscar-Winning Screenwriter." *Gizmodo*.
http://sploid.gizmodo.com/the-insane-creative-process-of-an-oscar-winning-screenw-
1601624702.

Karakaya, Bulut, C. Garcia, Daniel Rodriguez, M. Nityanandam, N. Labeikovsky, and T. Al Tamimi.
2009. "Sketch-It-Up! Demo." *Entertainment Computing–ICEC 2009*. Springer, 313–14.

Khaled, Rilla, Mark J. Nelson, and Pippin Barr. 2013. "Design Metaphors for Procedural Content
Generation in Games." *Proceedings of the SIGCHI Conference on Human Factors in Computing
Systems - CHI '13*. New York, New York, USA: ACM Press, 1509.
doi:10.1145/2470654.2466201. http://dl.acm.org/citation.cfm?doid=2470654.2466201.

Kilmas, Chris. 2009. "Twine."

Koebler, Jason. 2011. "Game Design Engages Students in STEM - US News." *US News*, July 25.
http://www.usnews.com/education/blogs/high-school-notes/2011/07/25/game-design-
engages-students-in-stem.

Koster, Raph. 2005. "A Grammar of Gameplay Game Atoms : Can Games Be Diagrammed ?" In
Game Developers' Conference.

Koster, Raph, and Inc Ebrary. 2005. *A Theory of Fun for Game Design*. Paraglyph press.
http://www.books4bestseller.com/theory-of-fun-for-game-design.pdf.

Kreimeier, Bernd. 2003. "Game Design Methods: A 2003 Survey." *Gamasutra*.
http://www.gamasutra.com/view/feature/2892/game_design_methods_a_2003_survey.php
#1.

Kuittinen, Jussi, and Jussi Holopainen. 2009. "Some Notes on the Nature of Game Design." *… in
Games, Play, …*. http://www.sosuaarhus-international.com/dokumenter/Game
literatur/Nature of Game Design.pdf.

Lantz, Frank. 2015. "Against Design." http://gamedesignadvance.com/?p=2930.

Lawson, Bryan. 2006. *How Designers Think: The Design Process Demystified*. *LONDON ARCHITECTURAL PRESS*. Vol. 3rd revise. Elsevier. doi:10.1007/s11060-008-9735-x. http://books.google.com/books?id=lPvqZJNAdG8C&pgis=1.

Leblanc, Marc. 2005. "Tools for Creating Dramatic Game Dynamics." In *The Game Design Reader A Rules of Play Anthology*, edited by K Salen and E Zimmerman. The MIT Press.

LeJacq, Yannick. 2012. "Something For Nothing: How The Videogame Industry Is Adapting To A 'Freemium' World." *International Business Times*, September.

Liapis, Antonios, Georgios N. Yannakakis, and Julian Togelius. 2013. "Sentient Sketchbook: Computer-Aided Game Level Authoring." *… on Foundations of Digital Games*. http://www.itu.dk/people/anli/mixedinitiative/sentient_sketchbook.pdf.

Literature and Latte. 2007. "Scrivener."

Löwgren, Jonas, and Erik Stolterman. 2004. *Thoughtful Interaction Design: A Design Perspective on Information Technology*. *Booksgooglecom*. http://www.loc.gov/catdir/toc/fy052/2004049891.html.

Lubart, Todd. 2005. "How Can Computers Be Partners in the Creative Process: Classification and Commentary on the Special Issue." *International Journal of Human Computer Studies* 63 (4-5 SPEC. ISS.): 365–69. doi:10.1016/j.ijhcs.2005.04.002.

Manker, Jon. 2011a. "Game Prototyping – Experiencing and Negotiating an Idea." *Design*.

———. 2011b. "Game Prototyping - The Negotiation of an Idea." In *Proceedings of DiGRA 2011 Conference: Think Design Play*, 4:1–16.

Marlin, Cyril. 2011. "Automated Testing: Building A Flexible Game Solver." *Gamasutra*, October. http://www.gamasutra.com/view/feature/134893/automated_testing_building_a_.php.

McEntee, Chris. 2012. "Rational Design :The Core of Rayman Origins." *Gamasutra*. http://gamasutra.com/view/feature/167214/rational_design_the_core_of_.php.

McGuire, Rory. 2006. "Paper Burns: Game Design With Agile Methodologies," June.
http://www.gamasutra.com/view/feature/131151/paper_burns_game_design_with_.php.

McMillan, Luke. 2013. "The Rational Design Handbook: An Intro to RLD." *Gamasutra*.
http://www.gamasutra.com/blogs/LukeMcMillan/20130806/197147/The_Rational_Design_H
andbook_An_Intro_to_RLD.php.

McWilliams, Laralyn. 2013. "The Metrics Aren't the Message." *Gamasutra*.
http://www.gamasutra.com/view/feature/188197/the_metrics_arent_the_message.php.

Natkin, S, and L Vega. 2004. "A Petri Net Model for Computer Games Analysis." *International
Journal of Intelligent Games Simulation* 3 (1): 37–44.

Nelson, Mark J. 2011. "Game Metrics without Players: Strategies for Understanding Game
Artifacts." In *Artificial Intelligence in the Game Design Process - Papers from the 2011 AIIDE
Workshop*, 14–18. http://www.scopus.com/inward/record.url?eid=2-s2.0-
84862684087&partnerID=40&md5=e71a4c9276adc25e668359a9beab5b18.

Nelson, Mark J., and Michael Mateas. 2008. "An Interactive Game-Design Assistant." In
Proceedings of the 13th International Conference on Intelligent User Interfaces, 90–98. ACM.

———. 2009. "A Requirements Analysis for Videogame Design Support Tools." In *Proceedings of
the 4th International Conference on Foundations of Digital Games*, 137–44. ACM.
http://portal.acm.org/citation.cfm?id=1536543.

Nevigo GMBH. 2011. "Articy:draft." http://www.nevigo.com/index.php?id=13.

Nummenmaa, Timo, Jussi Kuittinen, and Jussi Holopainen. 2009. "Simulation as a Game Design
Tool." *Proceedings of the International Conference on Advances in Computer Enterntainment
Technology - ACE '09*. New York, New York, USA: ACM Press, 232.
doi:10.1145/1690388.1690427. http://portal.acm.org/citation.cfm?doid=1690388.1690427.

Puckette, Miller. 1997. "Pure Data : Another Integrated Computer Music Environment."
Proceedings, Second Intercollege Computer Music Concerts, no. FEBRUARY 1970: 37–41.

Resnick, M, B Myers, and K Nakakoji. 2005. "Design Principles for Tools to Support Creative Thinking." http://repository.cmu.edu/cgi/viewcontent.cgi?article=1822&context=isr.

Reyno, Emanuel Montero, and José Á Carsí Cubel. 2009. "A Platform-Independent Model for Videogame Gameplay Specification." Edited by Atkins Barry, Kennedy Helen, and Krzywinska Tanya. *Breaking New Ground Innovation in Games Play Practice and Theory Proceedings of the 2009 Digital Games Research Association Conference*. Brunel University. http://www.digra.org/dl/display_html?chid=09287.28003.pdf.

Ries, Eric. 2011. *The Lean Startup*. *Working Paper*. doi:23. http://lean.st/.

Rollings, Andrew, and Ernest Adams. 2003. *Andrew Rollings and Ernest Adams on Game Design*. New Riders Games.

Rouse, Richard, and Steve Ogden. 2005. *Game Design : Theory & Practice*. *Design*. Wordware Pub.

Salen, Katie, and Eric Zimmerman. 2003. *Rules of Play: Game Design Fundamentals*. *MIT Press*. MIT Press. http://www.amazon.com/Rules-Play-Game-Design-Fundamentals/dp/0262240459.

———. 2005. *The Game Design Reader: A Rules of Play Anthology*. Edited by Katie Salen and Eric Zimmerman. *Review Literature And Arts Of The Americas*. The MIT Press. http://www.amazon.com/dp/0262195364.

Schell, J. 2008. *The Art of Game Design: A Book of Lenses*. *Annals of Physics*. Vol. 54. Morgan Kaufmann. Morgan Kaufmann.

Schön, Donald A. 1983. *The Reflective Practitioner*. *Pediatrics*. Vol. 116. Basic Books. doi:10.1542/peds.2005-0209. http://www.ncbi.nlm.nih.gov/pubmed/18796084.

———. 1992. "Designing as Reflective Conversation with the Materials of a Design Situation." *Knowledge-Based Systems* 3 (3). http://www.sciencedirect.com/science/article/pii/095070519290020G.

Shneiderman, B, and C Plaisant. 2006. "Strategies for Evaluating Information Visualization Tools: Multi-Dimensional in-Depth Long-Term Case Studies." In *2006 AVI Workshop on BEyond Time*

and Errors: Novel Evaluation Methods for Information Visualization.
http://dl.acm.org/citation.cfm?id=1168158.

Shneiderman, Ben, Gerhard Fischer, Mary Czerwinski, and Brad Myers. 2005. "NSF Workshop
Report on Creativity Support Tools." In .

Sicart, Miguel. 2008. "Defining Game Mechanics." *Game Studies* 8 (2): 1–18.
http://gamestudies.org/0802/articles/sicart.

Sigman, Tyler. 2005. "The Siren Song of the Paper Cutter: Tips and Tricks from the Trenches of
Paper Prototyping." *Gamasutra*.
http://www.gamasutra.com/view/feature/2403/the_siren_song_of_the_paper_.php.

Simon, Herbert A. 1969. *The Sciences of the Artificial*. *Cambridge, MA*. Vol. 1. MIT Press.
doi:10.1016/S0898-1221(97)82941-0.

Simon, Herbert A., and Allen Newell. 1972. *Human Problem Solving*. Prentice-Hall.

Sinclair, Brendan. 2013. "Devs Can't 'Create Magic' with a Spreadsheet in Front of Them."
Gamesindustry.biz. http://www.gamesindustry.biz/articles/2013-11-11-devs-cant-create-
magic-with-a-spreadsheet-in-front-of-them.

Smith, Adam M., Mark J. Nelson, and M. Mateas. 2009. "Computational Support for Play Testing
Game Sketches." In *Proceedings of the 5th Artificial Intelligence and Interactive Digital
Entertainment Conference (AIIDE)*.
http://www.aaai.org/ocs/index.php/AIIDE/AIIDE09/paper/viewFile/690/1061.

Smith, Adam M., Mark J. Nelson, and Michael Mateas. 2008. "Prototyping Games with BIPED."
Artificial Intelligence, no. 2007: 193–94.

———. 2010. "LUDOCORE: A Logical Game Engine for Modeling Videogames." In *Computational
Intelligence and Games (CIG), 2010 IEEE Symposium on*, 91–98. IEEE.
doi:10.1109/ITW.2010.5593368.
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5593368.

Smith, Gillian, J Whitehead, and M Mateas. 2011a. "Computers as Design Collaborators: Interacting with Mixed-Initiative Tools." *Proceedings of the Workshop on Semi-Automated ….* http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Computers+as+Design+Coll aborators+:+Interacting+with+Mixed+-+Initiative+Tools#0.

Smith, Gillian, Jim Whitehead, and Michael Mateas. 2011b. "Tanagra: Reactive Planning and Constraint Solving for Mixed-Initiative Level Design." *IEEE Transactions on Computational Intelligence and AI in Games* 3 (3): 201–15. doi:10.1109/TCIAIG.2011.2159716.

Sturt, George. 1923. *The Wheelwright's Shop*. Cambridge University Press.

Sutherland, I. E. 1964. "Sketchpad a Man-Machine Graphical Communication System." *Simulation* 2 (5): R − 3 − R − 20. doi:10.1177/003754976400200514.

Sweller, John, Jeroen J G van Merrienboer, and Fred G W C Paas. 1998. "Cognitive Architecture and Instructional Design." *Educational Psychology Review* 10 (3): 251–96. doi:10.1023/A:1022193728205.

Swink, Steve. 2009. *Game Feel*. *Game Feel*. Elsevier. doi:10.1016/B978-0-12-374328-2.00019-6. http://www.sciencedirect.com/science/article/pii/B9780123743282000196.

Sylvester, Tynan. 2005. "Decision-Based Gameplay Design." *Gamasutra*. http://www.gamasutra.com/view/feature/2264/decisionbased_gameplay_design.php.

Taylor, M J, D Gresty, and M Baskett. 2006. "Computer Game-Flow Design." *Computers in Entertainment* 4 (1). Association for Computing Machinery: 5. doi:10.1145/1111293.1111300. http://portal.acm.org/citation.cfm?doid=1111293.1111300.

Treanor, Mike, Bryan Blackford, Michael Mateas, and Ian Bogost. 2012. "Game-O-Matic: Generating Videogames That Represent Ideas." In *Proceedings of the Third Workshop on Procedural Content Generation in Games*. http://dl.acm.org/citation.cfm?id=2538537\nhttp://games.soe.ucsc.edu/sites/default/files/G ame-O-Matic (PCG2012).pdf.

Unity Technologies. 2005. "Unity."

Wetzel, Richard. 2014. "Introducing Pattern Cards for Mixed Reality Game Design." *Third Workshop on Design Patterns in Games*.

Whitson, Jennifer R. 2012. *Game Design by Numbers: Instrumental Play and the Quantitative Shift in the Digital Game Industry*. http://www.academia.edu/download/30941397/game_design_by_numbers-whitson-chapter1.pdf.

Winograd, Terry. 1996. *Bringing Design to Software*. Edited by Terry Winograd. *Journal of the American Society for Information Science*. Vol. 48. ACM Press Books. ACM Press. doi:10.1002/(SICI)1097-4571(199712)48:12<1149::AID-ASI10>3.0.CO;2-0. http://hci.stanford.edu/publications/bds/.

Yoyo Games. 1999. "GameMaker: Studio."

LUDOGRAPHY

Alone in the Park (2011, PC, iPad). Katharine Neil.

Candy Crush Saga (2012, Facebook, PC, mobile). King.

Cow Clicker (2010, Facebook). Ian Bogost.

Dota 2 (2013, PC). Valve Corporation.

Dear Esther (2012, PC). The Chinese Room.

Diner Dash (2004, PC, console, mobile). Gamelab.

Elder Scrolls (series of games). Besthesda Softworks.

FarmVille (2009, Facebook). Zynga.

Free Realms (2009, PlayStation 3, PC). Sony Online Entertainment San Diego.

Minecraft (2009, PC, others). Mojang.

Pokémon (1995-, GBA, DS). The Pokémon Company.

Portal (2007, PC, console). Valve Corporation.

Rollercoaster Tycoon (1999, PC). Chris Sawyer Productions.

Tomb Raider (1996, PC, console). Core Design.

Touch Detective (2006, DS). Beeworks.

APPENDIX

Progressimo description

Overview

Progressimo is a tool I developed. It is a progression design or planning (Butler et al. 2013) tool – a tool for planning the experience of player over the course of the game.

The designer builds a progression structure by creating **moments** and creating **connections** between those moments. The designer can constrain (lock or unlock) player progress via these connections, and also by gating access to moments (using **gates**) based on their content.

The content of moments comprises changes to the player's **progression state**. The progression state contains variable data that represents player progress, called **progression elements.**

The designer can verify and guide their design moves by creating **progression design rules** (constraints). These rules govern the use of progression elements within moments relative to progression state.

Progression units

Progression elements

Progression elements are any type of game content where the order in which that content is experienced over the course of the game matters. This can be, for example, skills that the player must learn, narrative knowledge that the player acquires, experience points or inventory acquired.

There are two main types of progression state elements. I have divided them into two types here not because they function differently in my tool, but to better explain how elements typically used in game design fit in to the concept of progression state elements and how this might affect the way they are treated by the designer within the tool. I have called these types "knowledge elements" and "game state variables".

Knowledge type elements can include skills or concepts that are introduced to the player (e.g. game mechanics, puzzle types, enemy types), access to areas of the game world that the player

has unlocked, and narrative developments. Knowledge elements are typically qualitative rather than quantitative in character, they will not be removed from the game state once added (i.e. they cannot be unlearned), and they feature in only part, not all, of the game.

Game state variables that are relevant to progression are usually expressed quantitatively and feature throughout the game. Common examples would be XP and currency.

Moments

Moments represent chunks of gameplay experienced over time by the player. As far as the tool is concerned, the function of a moment is to update game progress. It serves as a container for the introduction of, or quantitative changes to, progression state elements. Moments can be used to represent, for example, levels, tasks, missions or parts thereof.

Moments are unique (i.e. they are not modules that can be instanced). The designer can set moments to be either replayable or not replayable.

Moment elements

To define the content of a moment, the designer selects progression state elements from the **progression state elements list** and adds them to the **moment elements list** (see below).

The tool currently uses font styles to show whether progression state elements on the list are unlocked for use (i.e. satisfy **progression rules** – see below). A regular font style denotes that all paths to the selected moment contain the prerequisites for that element; italicisation means that one or more, but not all paths to the selected moment contain the prerequisites; a greyed out style indicates that it is not possible for the player to have encountered the prerequisites for that element before playing the selected moment.

As Figure rr shows, the designer can choose to include a progression element in a moment to signify its introduction or inclusion of as a knowledge element (e.g. "Stealth kill") or as a variable, specifying the number of instances of the element gained by completion of the moment (e.g. +3 rounds of ammunition, +10 XP). Negative quantities can also be used – for example, to specify the use of inventory items or currency that are used during the moment (e.g. a mission may cost -5 gold).

While the content of a moment is used by the tool for its own calculations, the designer may also want to use this list as a kind of "to do" or shopping list while working on the corresponding moment in a mission or level. Such a list features in *Refraction*'s tool's level editor component.

Element	Quantity
Stealth kill	1
Gold	-5
Ammo	3
XP	10

Figure rr: Moment elements list

Progression rules

Progression rules guide the designer vis-à-vis the order in which new game content (called "progression state elements" in the tool) may be introduced to the player over time. They are in the form of prerequisites, where one element is defined as a pre-requisite (or co-requisite) of another element.

Progression rules editor

Progression rules are defined in the **progression rules editor**. The progression rules editor allows the designer to create game elements and create progression rules by designating elements as prerequisites and co-requisites (logical disjunction ("or") relationships) of other elements. The editor offers two different interfaces for editing the rules.

Figure pp shows the matrix editor. This editor is functionally similar to constraints editor component of *Refraction*'s tool. The designer checks a cell on the grid to specify that an element in a column is a prerequisite of an element in a row. The tool automatically infers transitive prerequisites and visualizes these relationships in the matrix. Inferring these transitive relationships is a non-trivial calculation for a designer to perform manually. Dark blue cells indicate an explicit prerequisite, specified by the designer, while light blue cells indicate inferred ones.

The editor also displays co-requisite relationships (which do not feature in *Refraction*'s tool). These are indicated by aqua coloured cells. Currently these must be specified using the graph interface of the progression rules editor, shown in Figure qq.

Progression plan

The progression plan is where a game's progression structure is mapped out. To allow for the modelling of non-linear progression structures (such as the example shown in Figure 23), *Progressimo* has a graph-based editor. The designer creates a new moment by creating a graph node.
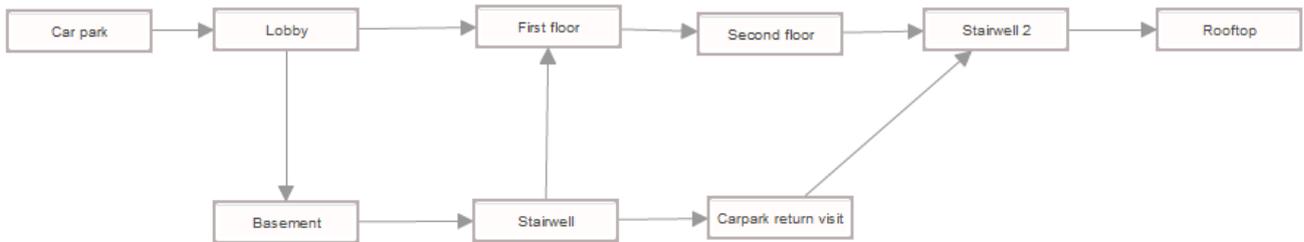

Figure ss: Example of a progression plan graph

Connections

Connections are one-directional graph edges that connect the moment nodes. They serve as a way to funnel and constrain progress. Access in the game from one node to another requires a connection.

For example, a completed stage in a mission unlocks the next mission stage, or completing a room or level allows the player to move to the next room or level. If the moment is set to **replayable**, it is assumed that the player may return and replay the content of the moment. If the **open-world structure checkbox** is selected (see below), moments that have no incoming connections can be accessed at any time (assuming that **gameplay gate** rules are satisfied).

There are two types of connections: **unlocking connections** and **prerequisite connections**.

Unlocking connections

The most typical connection is an **unlocking connection**. This connection denotes that the preceding moment has provided a sufficient condition to unlock the targeted moment. This means that if a moment has two preceding moments, completion of either of those moments will unlock the targeted moment.

Figure tt shows a level structure that contains unlocking connections. In this example, playing through a warehouse area (defined in the form of a Moment) unlocks access to both the street and sewers locations. The docks location can be accessed via either the street or the sewers.
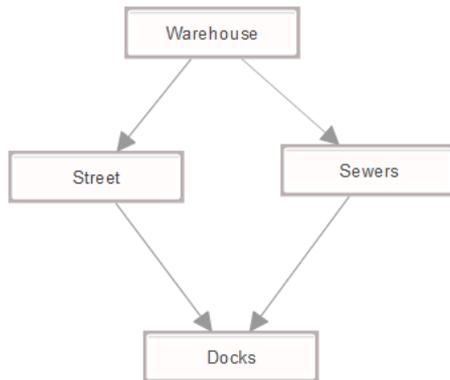
Figure tt: Mission graph with unlocking connections

Prerequisite connections

A **prerequisite connection** denotes that the previous moment must be completed in order to unlock the connected moment. This is useful when a set of mission tasks can be completed in any order. Prerequisite connections are visualised using edges with thicker line widths.
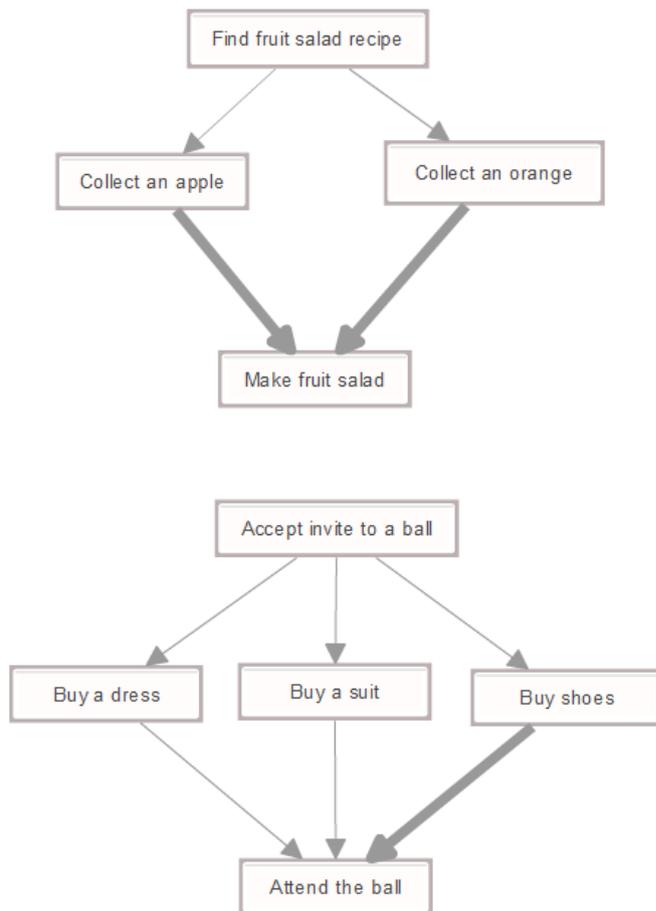
Figure uu: Two missions that use both *unlocking* and *prerequisite* (thick lines) connections

Figure 45 shows prerequisite connections. In the top mission, the player must collect both an apple and an orange before they can make a fruit salad. In the bottom mission, the player can either buy a dress or a suit to attend the ball, but shoes are compulsory.

Open-world structure checkbox

For open-world style games, or games where **gameplay gates** (see below) are the dominant device used for enforcing a progression structure (freemium games, for example, make heavy use of time and resource gates), the designer can check the **open-world inheritance** checkbox. This tells the tool that access from the end node of one connected group of moments to the start node of another group does not require a connection. In other words, a moment that has no incoming connections is accessible at any time (but may be gated).
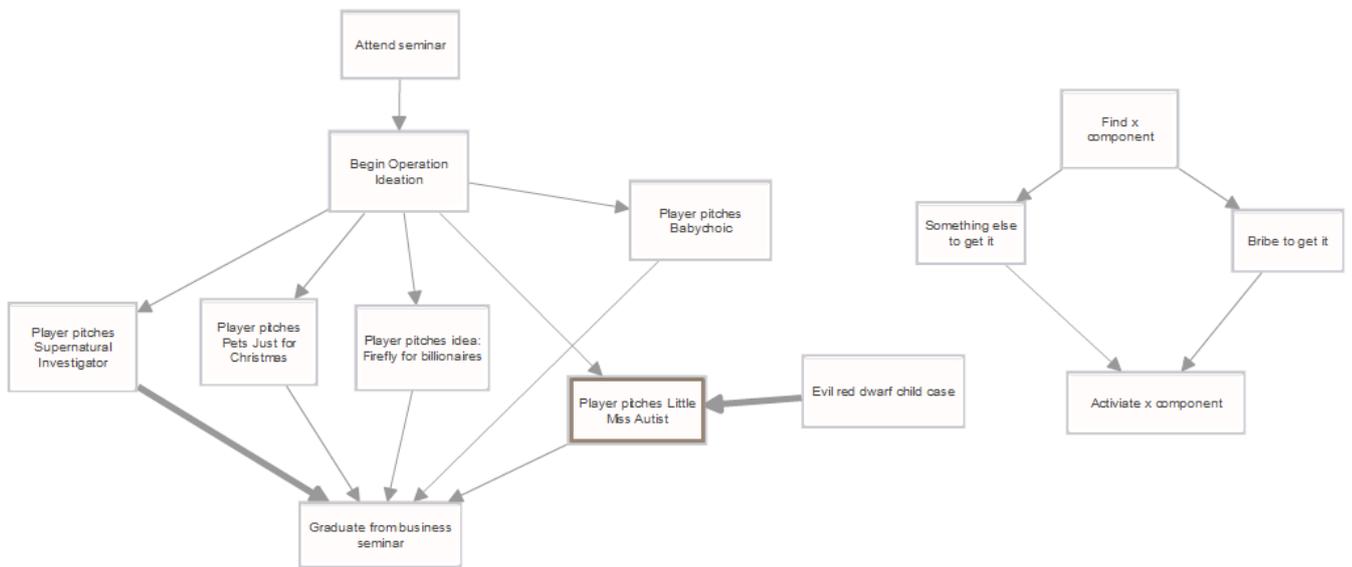
Figure vv: Two missions (from *The Particle Who Knew Too Much*) within an open world structure

Loopy Atoll

Design notes | Moment properties | Explore mode

☑ Open world inheritence

Progression elements

Collectibles
VIP passengers
Store
Mobile hazards
Unlock new things in the store
Static hazards
Speed boats
Reefs/sandbars
Spawning passengers delivered by plane (r
Sinking islands
Cargo
Coconut trees
Kava
untitled concept
Boat upgrades
Boat control
Loops

Paths to selected node: 1

Path: 14 16

Set inheritance depth

State updates   Add

Default | B

Concept	Quantity
Loops	1
Boat cooperation	1

Gameplay gates

Concept	Quantity

☐ Moment is replayable

Potential state

Element	Min	Max
Loops	1	1
Boat cooperation	2	2
Small boat	1	1
Medium boat	1	1

Explore

Recalculate content of selected moment

Loop practice

ap

Loopy Atoll

Five sist

2 boats practice

Coordination practice

Cooperation Islands

Coordination practice

Lupa Islands

Fafa islands

2 boats practice

Coordination practice

The Sandy Bottoms

Tenner Reef

Shipwreker's delights

Figure ww: Part of the progression plan for South Sea Trouble

Moment properties tab

When a moment node is selected its properties can be selected on the left hand side of the interface. These include a list of the progression elements that the moment contains, and a list of gameplay gates. Both these lists are editable. In addition, a list of all possible paths to the selected node is displayed, as well as the potential game state (in terms of progression elements) at the point that moment is reached.

Progression elements list

Alongside the graph interface a list of all game elements is displayed. The elements in the list are displayed differently according to their eligibility for use in the currently selected moment: eligible, ineligible and potentially eligible. Eligibility is based on the progression rules defined using the progression rules editor, and calculated using a graph traversal algorithm (my progression plan analyser) to determine whether a given element satisfies the rules applicable in the context of that moment. For example, if the rules stipulate that green ammo should not be included in the game before yellow ammo has been discovered, and no paths leading to the selected level contain yellow ammo, the "green ammo" element will be displayed as "ineligible". Conversely, if all paths leading to the level contain a yellow ammo element, the element will be marked "eligible".

The inclusion of a "potentially eligible" type is due to the non-linear progression structure. It services the case where one or more, but not all, paths leading to the level satisfy the rules associated with the listed element. Being "potentially eligible" may render the element appropriate or inappropriate for inclusion in the selected level, depending on the nature of the game or the element itself. The designer is left free to make an informed decision as to whether they wish to include the element.

Moment elements list

The **moment elements list** is the content of a moment, and represents the changes to the progression state as a result of the moment being played. Any element from the progression elements list may be added to the selected moment's progression elements list. The designer may then choose to give the element a numeric value, which can be negative (e.g. Gold -5).

Gameplay gates

A **gameplay gate** is imposed on the player during the game. The game state must meet the conditions of a moment's gates in order to either to begin or complete a moment.

Paths to selected moment list

The **paths to selected moment list** indicates all possible paths to the selected level. The total number of possible paths is displayed above the list. Selecting a path highlights all the moments along that path in the graph view. In Figure xx, the paths to selected level list can be seen on the bottom left. One of the paths is selected. The moments along the path are highlighted in the progression plan.

Figure xx: Paths to selected level. One path is highlighted.

251

Replayable moments

A checkbox allows the player to define whether the moment is replayable or not.

Potential progression state

This potential progression state list is shown at the bottom of the moment properties tab in Figure xx. A designer may wish to know which game elements the player has experienced or has potentially experienced by the time they reach the selected moment, based on the content of the moments that may or must be completed prior to it. The tool allows the designer to see this either for all possible paths to the level, or for a single path that they select in the paths to selected moment list. For example, based on all possible paths, the designer might see that the player has encountered a minimum 4 ammo pickups but potentially a maximum of 16. For a single path the minimum and maximum are identical.

If the *open-world structure checkbox* is checked, the potential state also takes into account the possibility that some or all of the moments unconnected to the selected moment could have been played.

Design notes tab

The design notes tab can be used by the designer to keep track of their design progress on a given moment. This is shown in Figure yy.

Design completion status

When the design notes tab is in view, the completion status on each moment is visualised in the progression plan graph. The outlines of nodes have a darker shade of grey according to their moment's design completion status.

This feature can be used to denote the state of design and/or production work on the moment within a game editor or level editing tool. This allows the graph view to be used as a kind of work plan, where the designer can quickly see which moments need to be worked on.

Notes

I include a field where the designer can write notes about the selected level. This could be brief text-based description of their intentions for the level, a "to do" list or issues to be resolved.

1_2b_intro_materials

| Design notes | Moment properties | Explore mode |

☐ Open world inheritence

Design progress

○ Not started

○ In progress

◉ Drafted

○ Ready for testing

Notes

Figure yy: The Notes tab

Explore mode tab

Switching to the explore mode tab activates **exploration mode** (see below).

Graph element visualisation

Refraction's tool includes a visualiser that plots the density and frequency of elements in the progression plan, in order to regulate progression considerations such as game pacing. In line with my approach, *Progressimo* computes this for the purpose of visualisation only: it highlights all nodes of the progression graph that contain a selected game element, thus affording the designer a two-dimensional overview of where instances of a given element are used in the game.

253

Highlighting a path (by selecting a path in the paths to selected moment list) allows the designer to see the number of instances of a selected element encountered between two points in the progression structure. The ability to visualise this in the form of a chart like *Refraction's* tool could be a feature added to this tool.

Exploration mode

When **exploration mode** is active, the designer can "play" the progression by clicking on accessible (unlocked) moments, step by step, in sequence. At each step where multiple moments are accessible, the designer makes a choice as to which moment to play. Access to moments is constrained by connections and gameplay gates. Figure zz shows the interface in explore mode with the progression plan on the right and the progression state information on the left.

Moment highlighting

Clicking on a moment in the progression plan "plays" the moment. Only unblocked moments can be played. When a moment is played the theoretical game state is updated using the progression elements contained in the moment (defined in the moment's elements list).

Once played, the moment's colour changes to beige – unless the moment is replayable, in which case the colour changes to blue. A moment that is unlocked and has never been played is coloured green.

Connected moments are then unlocked and highlighted green, as long gating conditions and progression rules satisfied. If the open-world structure checkbox is checked, all start nodes of connected moment groups that satisfy rules and conditions are also available.

Moments that are inaccessible via connections or blocked by gates remain white. If the designer clicks on a blocked moment they can view information on why that moment is currently blocked.

Moments that are accessible to the player (i.e. not blocked by gates and accessible via connections) but are blocked by design gates (i.e. their accessibility breaks progression rules) are highlighted in pink. This is a warning to the designer that they might want to make this moment inaccessible using connections or gates.

254

Figure zz: Progression state information in explore mode for *The Casimir Effect*

Progression state information

Progression state information is displayed in the explore mode tab.

Progression state

The **progression state** is a list of progression elements as variables. It is a simulation of what the game state might be at that point of the game, based on the sequence of moments that the designer has played to get to that point. Each time a moment is "played", the progression state is updated using that moment's elements list.

For example, a state that lists "Gold: 35" indicates that 35 gold would be in the player's inventory, based on their accumulation and expenditure of gold up until and including that moment. For knowledge type progression elements this can indicate how many times the player would have encountered that element. "Swimming: 5" could indicate that the player has had to swim in five moments so far.

Blockers

A moment being blocked means that game progression is not sufficiently advanced for the player to play the moment yet. In order words, the state does not contain the required progression elements in sufficient quantities to meet the conditions and rules that apply to that moment. As a consequence, the player is not able to play the blocked moment.

When the player selects a blocked moment they can view information on what is blocking progress to this element. There are three types of blockers: gameplay gates (as described above), **progress blockers** and design gates.

Gameplay gates

The gameplay gates list shows any gates for which the gating condition has not been satisfied. Gates that have been satisfied are not displayed.

Progress blockers

A progress blocker indicates that the progression state currently does not contain sufficient quantities of a progression element to meet the requirements of moment. Such requirements are expressed as negative progression element variables in the moment's elements list.

For example, a moment can result in an expenditure of a progression element – currency, for example. If an insufficient quantity of the progression state element – e.g. currency – is not present in the game state when that moment is reached during explore Mode, the tool effectively gates progress to this moment. Any such element will be displayed in the progress blockers list.

In this way, a progress blocker functions as a kind of implicit gameplay gating. As such it can serve as an alternative way to define a gate.

<u>Design gates</u>

Unlike gameplay gates, which gate access for the player to moments at run-time within the game, **design gates** enforce the progression rules on the designer at design time, within the tool.

Filtered notebook component

The notebook component allows the designer to add screenshots of a level design idea, which could take the form of a design pattern or even a finished level. The designer then tags the idea with one or more of the game elements it includes. When the designer comes to start work on a new game level they can use the **view only templates usable in selected level checkbox** to filter the notebook by the level's game elements list, to display all ideas that contain only the game elements relevant to that level.

Prepare for simultaneous attack
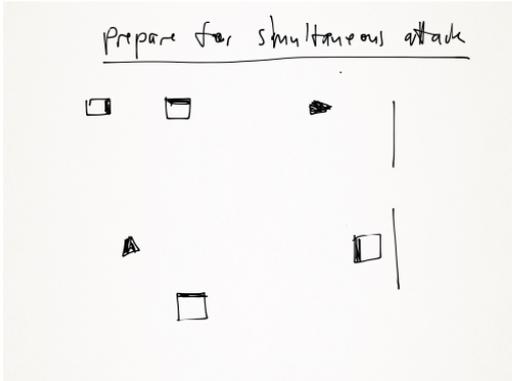
☑ View only templates usable in selected level

Save templates

Type C enemy	Single switch	IS THIS A DUPLICATE?	Shoot switch	Sequential switch
Simultaneous switch	Coloured switch	Explosions to trigger switch	Self-closing door
Turret	Bomb	Double material	Double happy	Coloured bullets
Coloured bomb	Colour mixing	Red bullet	Blue bullet	Purple bullet	Yellow bullet
Orange bullet	Green bullet	Red switch	Blue switch	Yellow switch	Purple switch
Orange switch	Green switch	Rainbow bullet	Red bullet lock	Blue bullet lock
Yellow bullet lock	Purple bullet lock	Orange bullet lock	Green bullet lock
Coloured double	Advanced combat	ammo	time camera	Type A enemy	Bullet lock
toggleable door	double door

New design template from image | Delete selected template | Clear concepts from selected template

Name	Concepts
Double with patrolling	Double material
Double material	Double material
Guarded material	Slide materialStealth
Create bomb to unblock	Bomb
3 simultaneous switches	Simultaneous switchShoot material
Create turret	Turret
Disable turret	TurretSlide material
Distance compensating for refire rate	Shoot enemy through material
Double happy	Double material
Double quad turret	Turret
Figure 8 turret	Turret
Hit switch	Hit switch
Quad turret	Turret
Sequential switches	Sequential switch
Simultaneous switches	Simultaneous switch

Figure aaa: Filtered notebook component

258

RESUME LONG

Introduction

Depuis plus d'une décennie, les grands créateurs de l'industrie du jeu vidéo tout comme les chercheurs ont sporadiquement mais régulièrement identifié, à la fois comme un problème et comme une opportunité, le manque d'outils de conception[32] mis à disposition des concepteurs de jeux (Game Designer). Ils ont fait valoir que, si l'objectif de la conception du jeu vidéo est d'atteindre de hauts niveaux de créativité et d'expression, des outils formels pour la conception de jeux vidéos (que ce soient des modèles conceptuels ou des logiciels) doivent être développés.

Dans ce but, un travail théorique a été entrepris pour extraire des techniques de conception actuelles des principes traduits sous la forme de modèles formels, de façon à créer des taxonomies et à développer des systèmes de notation graphique (Björk et Holopainen 2005; Koster 2005; Araújo et Roque 2009; Natkin et Vega 2004; Bura 2006; Sicart 2008; Reyno et Cubel 2009; Cook 2007). Plus récemment, quelques tentatives ont été menées par les chercheurs pour concrétiser certaines de ces idées dans des outils logiciels (Dormans 2009; A. M. Smith, Nelson, et Mateas 2008; Karakaya et al. 2009).

Cependant la communauté des praticiens a largement ignoré ce travail, laissant sans réponse la question de savoir si les outils de conception peuvent utilement soutenir et développer la pratique de la conception du jeu. Même au sein de la communauté scientifique, relativement peu de discussions ou de critiques comparatives mettent en valeur ce domaine émergent.

Nous estimons que ce débat et le développement du travail de création d'outils pour le Game Design doit s'appuyer sur une évaluation pratique des méthodes et des outils proposés dans la littérature. Notre projet de recherche doctorale vise à contribuer à cette démarche. Notre point de vue est celui d'une praticienne. Cette étude d'outils de conception de jeux est donc fondée sur la construction d'une méthode d'évaluation des pratiques menée en tant que participante-

---

[32] Dans ce texte, le terme de design a été traduit par « conception » et le terme de designer par « concepteur », qu'il faut considérer comme un terme grammaticalement neutre et non comme un masculin.

observatrice, en appliquant un ensemble d'outils de conception de jeux à un ensemble de cas d'études.

Nous avons examiné spécifiquement la conception de jeux vidéo, bien que la plupart des outils de conception de jeux vidéo puissent être également utiles aux les jeux de société. En outre, en raison du cadre de ce travail, un projet de recherche doctorale, nos études de cas reposent sur une équipe de conception et de développement réduite à une seule personne sauf en ce qui concerne une des études de cas qui a impliquée également un concepteur graphique. De ce fait, nous avons dû en déduire la forme que pourrait prendre les expériences de conception dans le cadre d'un travail en grandes équipes et dans des contextes de développement plus industriels.

La méthode d'analyse utilisée pour comparer les cas étudiés s'appuie sur différentes disciplines de recherche. Tout d'abord, le travail qui a été fait sur le thème des outils de conception de jeux s'inscrit directement dans les disciplines des *Game Studies*, de l'informatique et, plus particulièrement de l'interaction Homme-Machine. Plus spécifiquement nous nous appuyons sur des connaissances issues de domaines où la discussion sur les outils et leurs relations avec la conception pratique est la plus avancée : les études de la conception (*Design Studies*), de recherche d'outil de conception et d'outil de soutien à la créativité. Afin de comprendre certains des défis auxquels sont confrontés spécifiquement par les concepteurs de jeux, nous considérons également les aspects de la culture, de la pratique de la conception de jeux, ainsi que des facteurs contextuels liés aux tendances actuelles de l'industrie du jeu qui influent sur la conception des jeux.

Le processus et les activités de conception de jeux vidéo

Les outils de conception apportent un soutien à la pratique de la conception de jeux. Dans ce chapitre, nous passons en revue ce qu'est cette pratique aujourd'hui.

Tout d'abord, nous donnons un état de l'art portant sur les activités et les processus de conception de jeu, domaine qui est, pour l'instant peu documenté. Nous analysons ensuite le processus de conception tel qu'il est pratiqué, mettant en évidence les activités, leur enchaînement et les productions successives qui mènent progressivement à la bible du jeu. Nous accordons une

attention particulière aux tendances et aux orientations actuelles : la conception axée sur les données et la « culture du *game jam* ».

Nous analysons les différentes tâches de conception. Par tâches que n'entendons pas ici le type d'activités de conception réalisées, mais plutôt les parties de la situation de la conception du jeu en cours d'élaboration : développement du concept (idéation), conception de la progression, conception des missions/niveaux de jeu, conception du système et équilibrage.

Nous soutenons que, étant donné que les concepteurs de jeux n'ont pas de soutien méthodologique ni d'outil pour penser la conception, il n'est pas surprenant de voir la pratique devenir fortement tributaire du recueil de gros volumes de données portant sur les comportement des joueurs, d'algorithmes d'analyse statistique et de l'interprétation heuristique des résultats.

La conception du jeu en tant que pratique

Ce chapitre examine certains concepts proposés de façon générale par les études des processus de conception (Design) pour les tenter de les appliquer au domaine du jeu vidéo. En outre, ces concepts fournissent un vocabulaire et une boîte à outils utilisés pour donner un cadre à la discussion de nos travaux pratiques dans les chapitres 7, 8 et 9.

Le principal problème mis en évidence est que, dans le domaine des jeux, une partie importante des décisions de conception sont prises pendant la phase de fabrication (développement) du jeu. Le jeu est conçu par essais successifs ce qui nécessite, pour l'expérimenter une version prototypée ou en développement. Cette approche, qui est considérée comme incontournable dans la pratique, rapproche la conception d'un jeu d'un artisanat (ce que Christopher Alexander nomme *unselfconscious design* (Alexander 1964), soit littéralement « conception qui n'a pas conscience d'elle-même », et que nous traduirions par « conception qui se fait sans recul sur elle-même »).

Dans le domaine des jeux, nous manquons donc de moyens pour séparer la conception du développement. Pour mieux mettre en évidence et faciliter l'émergence d'activités de conception proprement dites, un concepteur doit être en mesure de formaliser et matérialiser le processus et les productions de son travail afin que la réflexion de conception puisse porter sur cette

représentation et non - sur une version jouable du jeu comme cela se produit le plus souvent aujourd'hui.

Pour représenter la situation de conception distincte de la fabrication, un concepteur utilise ce qu'on appelle, dans la théorie des processus de conception, « un dessin de conception » (*design drawing*). Nous manquons de dessins de conception pour les jeux. Ce manque est probablement en grande partie attribuable à la nature des jeux elle-même : la conception de jeu est donc un problème du « second ordre », où le concepteur n'intervient qu'indirectement sur le résultat de son travail (Salen et Zimmerman 2003, 168).

Pour satisfaire le besoin de séparer la conception du développement, des tentatives pour créer des d'outils de conception de jeu ont été menées. Nous les présentons dans la dernière partie de ce chapitre. Nous décrivons le contexte et l'historique, à la fois dans le milieu universitaire et dans l'industrie, des travaux de Recherche et Développement portant sur les outils de conception de jeu.

Ces outils offrent aux Game Designers un moyen pour dessiner la conception : représenter et manipuler la « situation de conception ». Voir les outils de conception de jeu sous cet angle, à savoir que le dessin de conception, pourrait permettre à la pratique de la conception de jeux de passer d'une pratique artisanale à la conception réflexive. Ceci montre l'importance des travaux sur les méthodes et outils de conception de jeu, ainsi que la nature qualitative du changement que l'adoption de ces outils dans la pratique engendrerait.

Outils de conception de jeu

Pour les besoins de notre recherche, nous définissons un « outil de conception de jeux » comme suit : outil logiciel ou conceptuel dont la fonction principale est d'apporter un soutien aux concepteurs de jeux dans la pensée de conception.

Dans ce chapitre, nous analysons chaque élément de cette définition, que nous décrivons en détail. Nous examinons ensuite les outils que la communauté de la recherche et que l'industrie ont produit et qui entrent dans ce cadre. Nous comparons ensuite et discutons les outils en relation avec les types d'activités de conception qu'ils soutiennent.

Le choix des outils utilisés a été en grande partie fondé sur leur disponibilité. Les outils de réflexion dynamique (*brainstorming*) ne sont pas considérés. Dans cette étude nous utilisons les outils suivants.

Outil de modélisation du système:

-   *Machinations* (Dormans 2009) (voir Figure) offre un moyen de communication et de mo-délisation de "la structure des systèmes de jeu et les structures répétitives (patterns) qui pourraient être trouvés dans ces structures". La représentation sous forme de diagrammes, inspirés des réseaux de Petri, permet de représenter la dynamique et de la simuler.

Outil de conception narrative:

-   *Articy : Draft* (Nevigo GMBH 2011) Figure) est un outil commercial destiné au design des jeux narratifs, par exemple des jeux d'aventure ou des jeux de rôle. Une interface program-mation graphique permet la conception des éléments de design de jeu à choix arborescent (enchainement, dialogue ...). Il propose également une interface graphique pour l'édition et l'organisation de données associées aux entités de jeu comme des PNJ. De plus, il fournit à un éditeur pour ordonner et simuler l'enchainement des niveaux de jeu.

Outils de conception mixte d'initiative. Le design mixte d'initiative est une approche actuellement favorisée par des chercheurs dans le domaine de génération de contenu procédurale pour des jeux. Le but est d'améliorer les techniques de génération procédurales actuelles qui permettent peu de contrôle au concepteur. "Mixte" désigne la participation tant de l'ordinateur que du

concepteur, le couple entrant dans un dialogue de conception (G. Smith, Whitehead, et Mateas 2011).

- *Ludoscope* (Dormans 2010) (voir Figure) est utilisé pour la conception de la progression : la structure de missions (par exemple des tâches et des événements) et la structure "de l'espace", l'espace ou le monde dans lequel navigue dans le joueur. Dormans fournit une solution de la question du contrôle et la communication entre le concepteur et le système de génération de contenu en utilisant des grammaires formelles et des règles de réécriture.

The Sentient Sketchbook (Carnet à croquis doué) (Liapis, Yannakakis, et Togelius 2013) (voir Figure j) fournit l'assistance pour le level design de jeux de stratégie. Il permet au concepteur d'assurer la jouabilité de la conception spatiale du niveau et a pour but de réduire l'effort impliqué dans la production de cartes détaillées.

Outils de conception de la progression. La notion de progression dans un jeu recouvre plusieurs aspects. Il s'agit par exemple de l'acquisition des compétences par le joueur et, corrélativement, de la progression de la difficulté des défis qui lui sont proposés. Mais cette progression est elle-même liée à la structure dynamique du jeu (niveaux successifs), à la structure de l'espace de jeu (qui détermine les trajets possibles des entités contrôlées par le joueur) et éventuellement à une progression narrative.

- *Skill chains* (Cook 2007) Le but d'un *skill chain* (chaîne de compétences) est de faciliter la décision sur les compétences que le joueur doit progressivement apprendre et l'ordre dans lequel ces compétences doivent être acquises.
- « Planification de la progression » (dans l'outil d'édition du jeu *Refraction*) (Butler et al. 2013) (voir Figure m) facilite la conception de la progression du jeu, basée sur des règles qui contrôlent l'ordre dans lequel les concepts sont introduits dans le jeu.
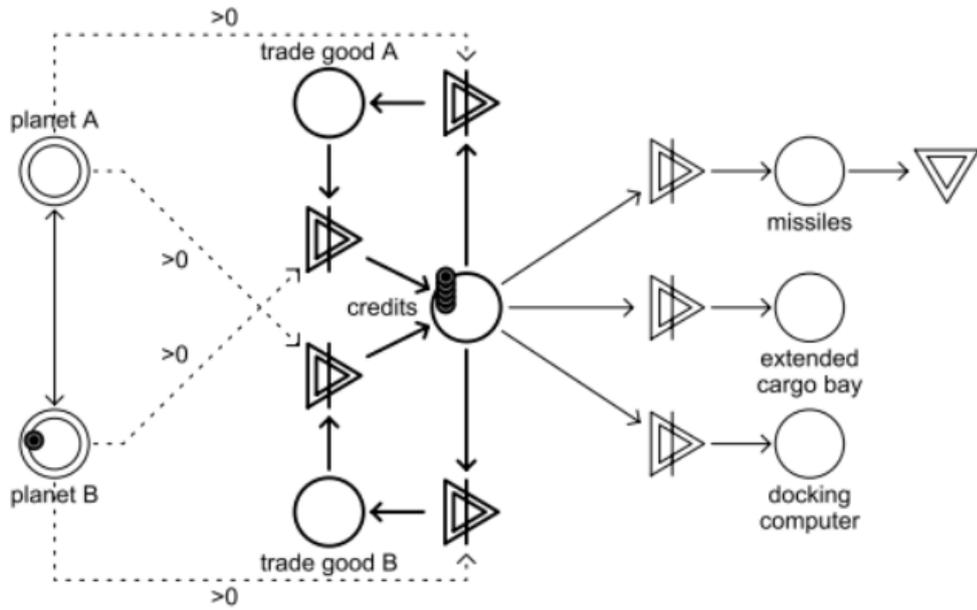
Figure 1 : diagramme du jeu *Elite* créé avec *Machinations*
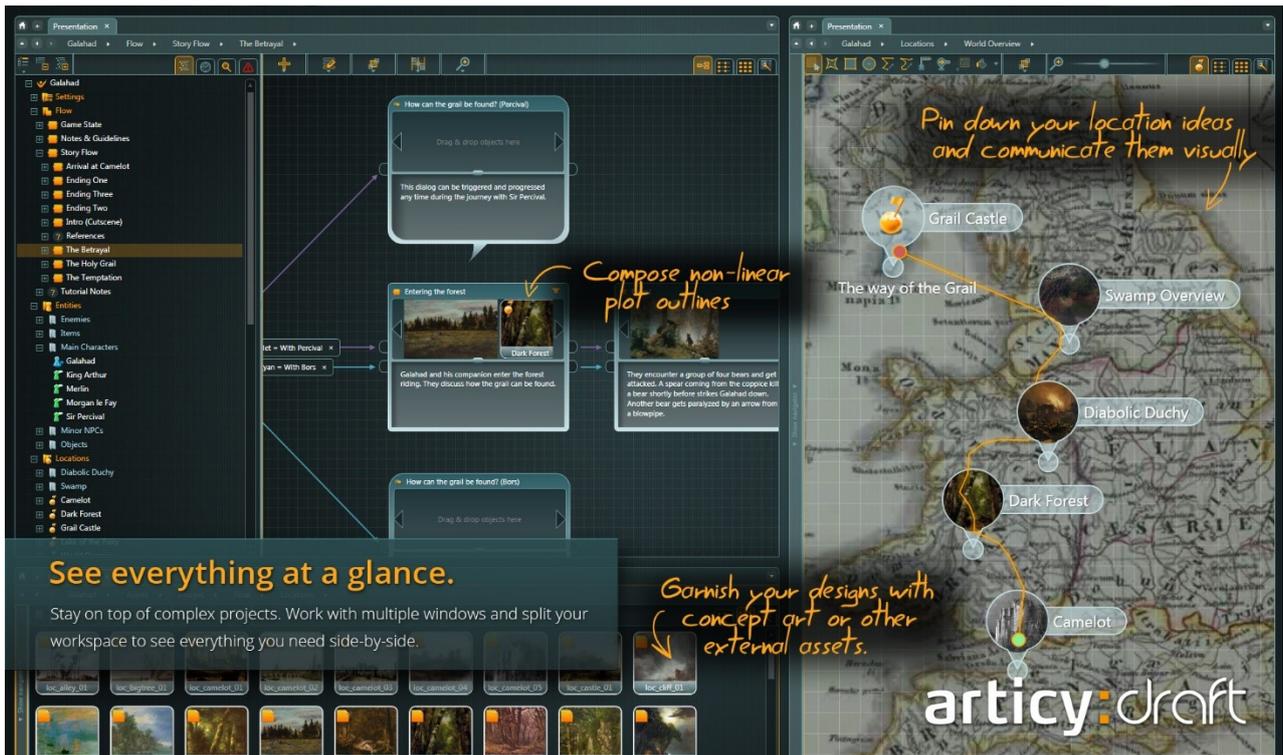
Figure 2 : capture d'écran annotée d'Articy:Draft
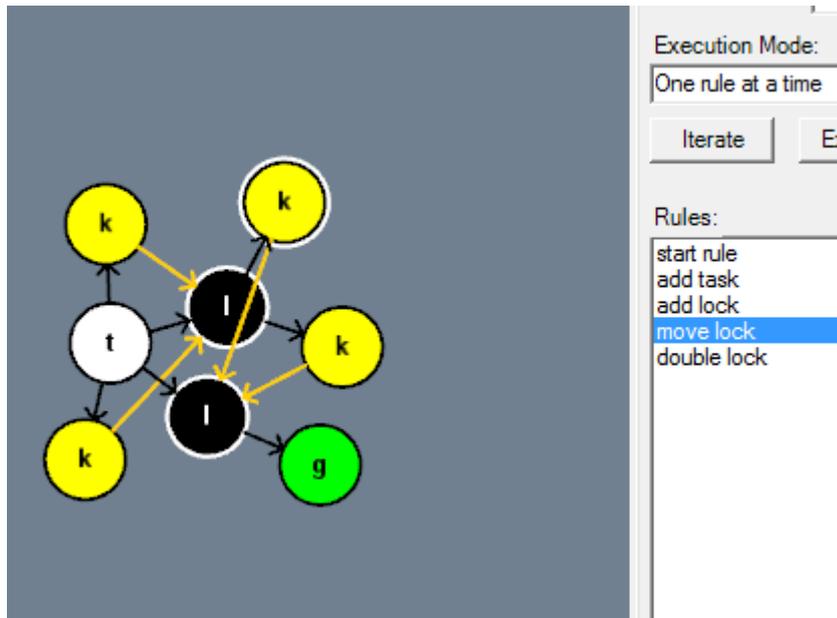
Figure 3 : diagramme d'une mission créé avec *Ludoscope*
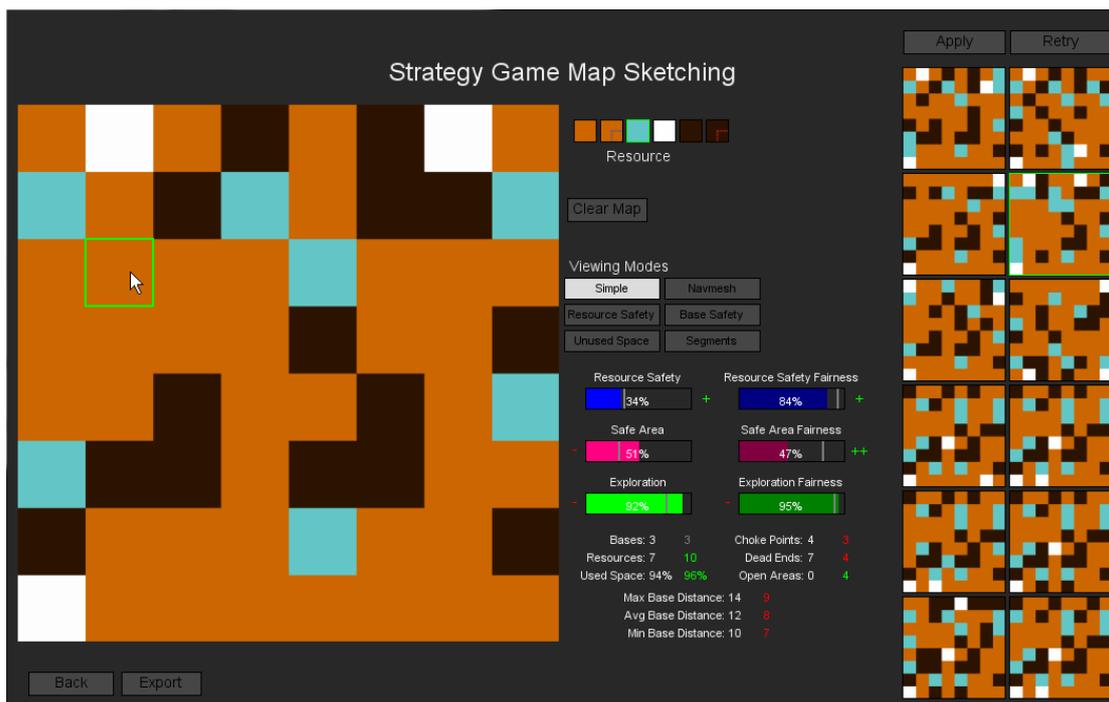
Figure 4 : édition d'une carte de jeu avec *The Sentient Sketchbook*
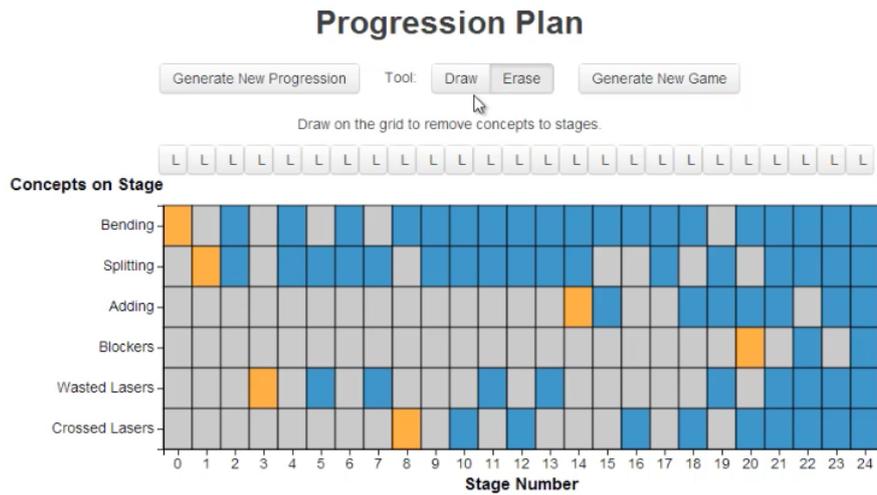
Figure 5 : le module de plan de progression dans l'outil d'édition du jeu *Refraction*

Le tableau 1 offre une synthèse comparative des outils de conception de jeu, et les met en relation avec les activités de conception auxquelles ils apportent un soutien. En outre, en utilisant une clé de lecture spécifique au domaine, il indique les types de tâches de conception, au sens précisé au chapitre 2, auxquelles ils s'appliquent de façon indubitable.

Tableau 1 : Comparaison des activités de conception et de tâches dans les outils de conception de jeux

|	Les activités de conception									Tâches de conception de jeux			
	Génération de matériaux (contenu du jeu)	Produire ou transformer des matériaux (changement persistant)	Donner des informations	Contraintes de vérification ou validation	Back talk explicite	Simulation	Simulation interactive	Communication	Exploration	Développement du concept (idéation)	Design de progression	Conception de niveau/mission	Conception du système et équilibrage
Machinations		x	x			x	x	x	x	x	x		x
Planification de progression (au sein de l'outil d'édition *Refraction*)		x	x	x				x			x		
The Sentient Sketchbook	x			x								x	
Ludoscope	x	x							x			x	
Game design patterns										x			x
Skill chains		x										x	
Tanagra	x				x							x	
Sketch-it-up!						x		x		x			
Ludocore + BIPED					x	x	x					x	x
Articy:Draft	x	x						x			x	x	

Les questions de recherche et de la méthode

Ce chapitre explicite la motivation qui sous-tend ce projet et sa contribution au domaine de la recherche. Nous explicitons nos questions de recherche et répertorions les outils traités dans cette étude, ainsi que la méthode de recherche utilisée et le raisonnement qui nous a conduite à ce choix.

Problématique

Nous avons cherché à répondre aux questions de recherche suivantes :

1) Les outils de conception de jeu peuvent-ils améliorer la pratique du Game Design?
2) Les outils de conception de jeu peuvent-ils élargir la portée des résultats du Game Design?
3) Comment les outils ont-ils un impact sur le processus de conception du jeu?

Plus généralement, nous visons à produire les connaissances qui peuvent contribuer au développement d'outils de conception de jeux.

La méthode

Il nous semble nécessaire pour faire avancer ce domaine de recherche de dépasser de la simple validation d'outils et de réaliser: une évaluation par des tiers, comparative et pratique des outils de conception de jeu.

Une enquête d'intérêt et d'usage parait appropriée pour cette étude, semble correspondre aux paramètres de notre projet et à nos domaines d'expertise en tant que praticienne-chercheuse. Notre objectif n'est pas de fournir des réponses définitives, mais bien plutôt de compléter d'autres études, en offrant une pièce supplémentaire du puzzle ou une seule ligne parmi plusieurs « lignes de preuves convergentes ».

Notre enquête est fondée sur la pratique, elle utilise pour cela une méthode d'observation participante en tant que praticienne réflexive. Nous avons utilisé toute une gamme d'outils dans le cadre du travail de conception sur un groupe d'études de cas de conception de jeux. Nous avons également utilisé des méthodes de travail de conception classiques et des outils (prototypage, documentation, etc.) où ils semblent les plus appropriés, en les adaptant le cas échéant.

Vue d'ensemble des études de cas

Nous avons conçus et construits plusieurs jeux dans le but d'analyser le fonctionnement des outils de conception de jeu face à des problèmes de conception réalistes. Le chapitre fournit le contexte ces analyses : il donne une brève description du design final de chaque projet. Nous résumons également chaque processus de conception. Les éléments brièvement présentés ici, sont décrits et expliqués plus en détail dans les sections suivantes où nous discutons de notre expérience d'usage des différents outils.

 Les jeux présentés sont:

The Casimir Effect
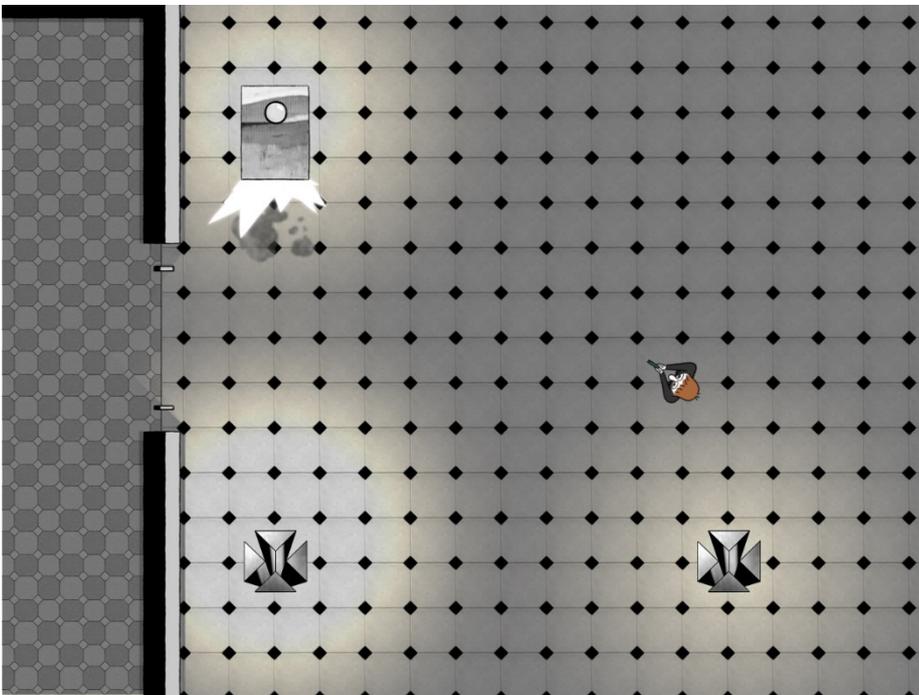
Figure 6 : capture d'écran de *The Casimir Effect*

The Casimir Effect (voir Figure 6) est un jeu de tir en vue de dessus au mécanisme original composé de combat et de puzzle. Le combat se fait à distance, avec une seule arme et un seul projectile. Le tir classique est possible mais le joueur a la possibilité d'un tir indirect via l'utilisation des cubes, positionnés dans l'environnement, qui amplifient les dégâts du projectile, modifient sa trajectoire, et permettent au joueur de stocker et de rythmer des attaques simultanées et multidirectionnelles*.*

The Particle Who Knew Too Much

The Particle Who Knew Too Much est un jeu d'aventure conçu au sein d'un système de jeu d'aventure hybride texte/graphique que j'ai conçu pour un jeu antérieur appelé *Alone in the Park* (2011). Dans ce système, le gameplay point-and-click conventionnel est remplacé par un système combiné de conversation et d'inventaire tandis que l'action du jeu est représentée par le texte généré en temps réel en réponse aux actions du joueur sur la carte 2D, du monde du jeu.

South Sea Trouble
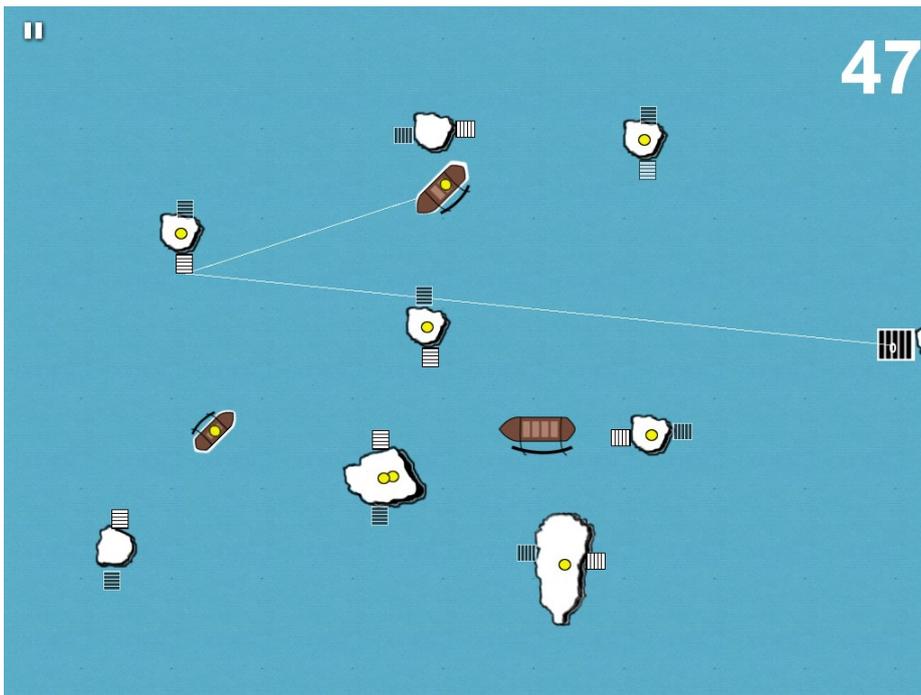
Figure 7 : capture d'écran de *South Sea Trouble*

South Sea Trouble (voir Figure) est un jeu de puzzle orienté stratégie dans lequel le joueur coordonne une petite flotte de bateaux, aux tailles et vitesses variables, pour prendre et déposer des passagers entre différentes îles en temps limité. Bien que le jeu soit en temps réel, un mécanisme, que j'appelle « commande - file d'attente » (*command-queueing*) permet un ressenti de jeu très similaire à un gameplay planification-exécution (*plan-and-execute*).

Le joueur débloque progressivement un enchainement assez souple de niveaux, où chaque niveau est une carte de la taille de l'écran, présentant une distribution spécifique d'îles, de récifs et de bateaux. L'enchaînement des niveaux est ramifié assez souplement, afin que tous les niveaux n'aient pas besoin d'être terminés pour que le joueur progresse. Chaque niveau est introduit par

271

un bref récit qui offre une logique fictive expliquant l'entreprise. Ces récits s'insèrent dans une narration globale.

Loot the Room

Figure 8 : capture d'écran de *Loot the Room*

Loot the Room (voir Figure 8) est un jeu de puzzle-stratégie. Esthétiquement, le jeu pourrait être décrit comme un donjon-crawler accéléré et *casual*. Le joueur dirige une équipe (des pillards vikings) à travers un espace au style de donjon labyrinthique (un monastère) pour piller les trésors des différentes salles. Les salles sont de tailles différentes, selon le nombre de pillards qui doivent s'y trouver simultanément afin que la salle puisse être pillée. Le joueur envoie des personnages dans le donjon à plusieurs reprises sur des itinéraires différents, en essayant de coordonner les actions de pillage des différents personnages. Les efforts du joueur permettent de contourner différents obstacles et de récupérer des bonus tout en résolvant le niveau dans son ensemble. Le joueur contrôle le jeu très simplement en traçant un itinéraire dans le labyrinthe pour chaque personnage.

Ultraworm

Figure 9 : capture d'écran d'*Ultraworm*

Ultraworm (voir Figure) est un jeu de puzzle en temps réel où le joueur contrôle une flotte de véhicules afin de collecter et de livrer des ressources de différentes couleurs à leurs destinations de couleur similaire. Les ressources apparaissent spontanément au hasard, à des emplacements inoccupés et selon un rythme croissant progressivement. Débarrasser un emplacement de sa ressource et la livrer à sa destination rapporte un point au joueur, l'objectif étant de marquer autant de points que possible avant la fin de la partie. Celle-ci survient quand il n'y a plus d'emplacements libres ou une ressource pourrait apparaître Du point de vue de la grille de lecture de *Patterns in Game Design* (Björk et Holopainen 2005), *Ultraworm* est un « jeu qu'on ne peut gagner », où la difficulté augmente continuellement jusqu'à ce que le joueur fasse une erreur.

Les cinq études de cas montrent principalement les besoins et la finalité propres à la conception de niveau (*level design* ou *mission design*), ainsi que certaines exigences de la conception du système de jeu, et de la narration. La nature de ces besoins diffère en fonction de types de jeux évoqués.

Notre approche méthodologique est guidée par les impératifs de la conception, malgré qu'ils soient souvent imprévisibles, plutôt que de laisser les besoins de l'évaluation influencer la méthode. Ceci a transformé le processus de conception lui-même en un processus de découverte, révélant naturellement les tâches qu'un outil donné peut réaliser.

Certaines tâches de conception ont été plus fréquentes que d'autres, avec pour conséquences une utilisation plus importante de certains outils que d'autres. Par exemple, nous avons constaté que nous avions trouvé un usage pour la *progression planning* (planification de progression) pour presque tous les jeux en dépit de la variété des genres. A l'inverse, *The Sentient Sketchbook* vise une tâche très spécifique pour une fonction particulière dans un genre particulier de jeu. Nous avons été incapable finalement, de l'explorer en profondeur. Comme la plupart des genres de jeux ont une certaine forme d'aspects narratifs, nous avons pu utiliser *Articy:Draft* pour la plupart des études de cas. Nos études de cas ne nécessitent pas autant de travail de conception à l'échelle du système que ceux que l'on trouve dans certains types de jeux vidéo, de simulation, de stratégie ou certains jeux de société. En conséquence, même si nous avons pu utiliser *Machinations* pour certaines tâches de conception système, notre travail d'évaluation est resté limité.

Observation et analyse partie 1: La forme de représentation

Ceci est le premier des trois chapitres dans lesquels nous analysons et discutons certaines de nos expériences d'utilisation d'outils de conception de jeux. La discussion s'articule autour des thèmes et des critères d'évaluation définis dans la littérature sur les outils de conception et les outils de soutien à la créativité.

Dans le chapitre 3, nous avons fait valoir que la principale faiblesse dans la pratique de la conception de jeu est l'incapacité à représenter la situation de conception. En conséquence, la question de l'utilité d'une représentation fournie par les outils de conception devient l'élément central de notre évaluation.

L'une des fonctions clés d'un outil de conception est de permettre au concepteur de créer une représentation d'une situation de conception. Dans ce chapitre, nous examinons la forme que cette représentation peut prendre. Nous discutons de la représentation visuelle, du langage utilisé par un outil et de son niveau d'abstraction, de l'exhaustivité de la représentation qui en résulte et des ajouts et des modifications de la pensée de conception que la représentation demande au concepteur.

Nous proposons cinq observations issues de nos expériences. Celles-ci décrivent : les représentations visuelles d'une situation de conception de jeu en utilisant *Articy:Draft*, le

déplacement de l'attention de la mécanique à la dynamique au sein de la représentation (voir Figure 10), avec *Machinations,* le travail simultané sur plusieurs couches grâce à la génération du contenu procédurale (*procedural content generation*), l'imbrication pour faciliter le travail à haut niveau dans *Articy:Draft* , la suppression des lacunes dans la représentation de l'outil grâce à la documentation, des croquis et des prototypes.
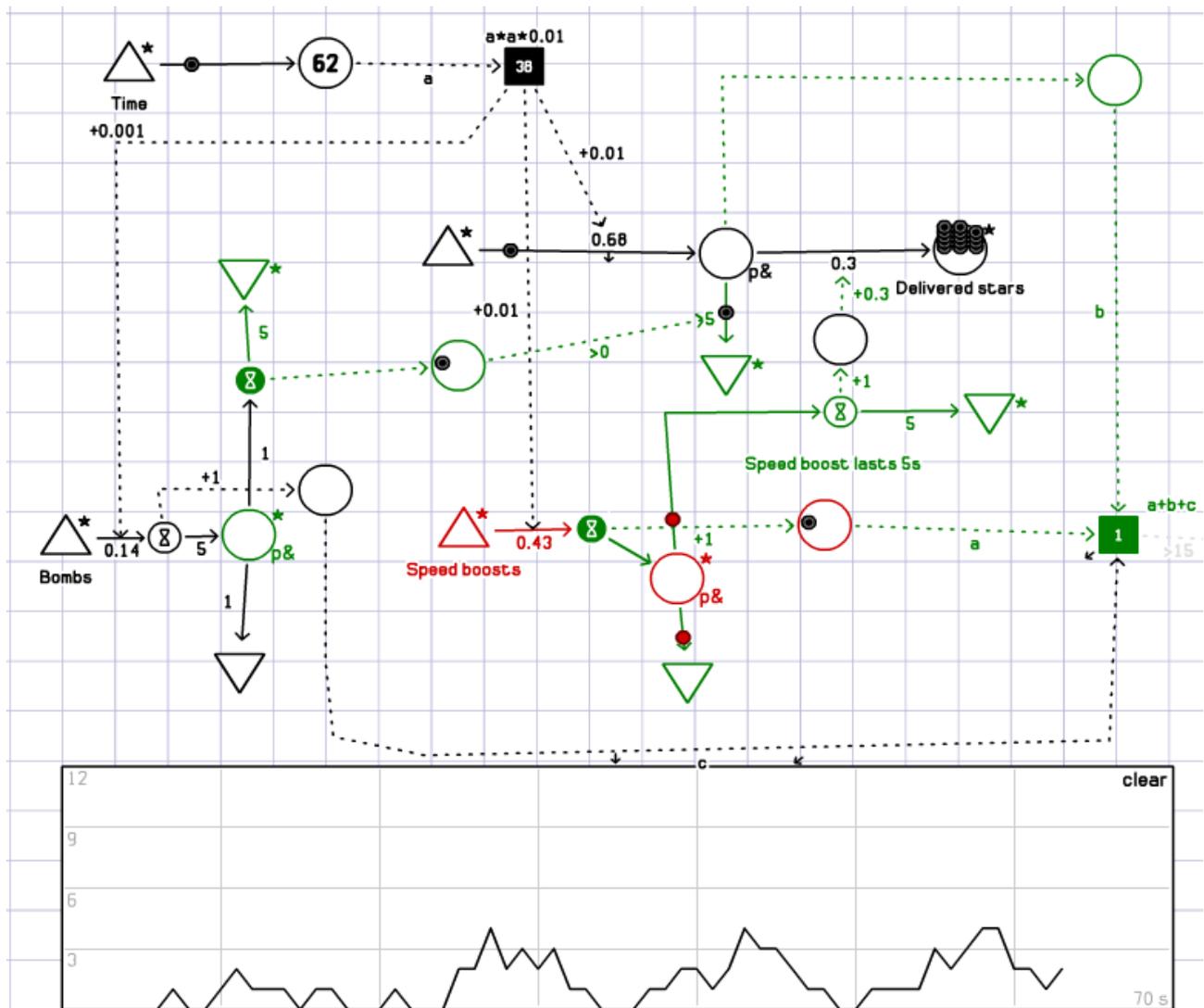
Figure 10 : la dynamique (en bas) simulé de la mécanique (en haut) de *Ultraworm* dans *Machinations*

Observation et analyse partie 2: La représentation

Ce chapitre se penche sur la représentation d'une situation de conception et la capacité de faire des choix avec des outils de conception de jeu. Nous discutons ensuite des coûts, des contraintes

liés à cette représentation et des compétences demandées au Game Designer. Ceci comprend une discussion sur les exigences de prise en main des outils et la capacité des outils à favoriser un usage créatif. Enfin, ce chapitre traite de l'acte consistant à utiliser la représentation pour identifier les problèmes et les alternatives à un choix.

Nous concluons sur trois observations : comment l'espace virtuel nous a aidée à organiser nos idées, l'utilité d'une interface dédiée aux annotations, et comment nous nous avons été amené à utiliser du contenu généré de manière procédurale.

Observation et analyse partie 3: Intégration des outils de conception de jeu en pratique

Nous examinons le contexte d'usage des outils et considérons ces outils de conception du point de vue de leur intégration dans la pratique de la conception de jeux, existants et futurs.

Tout d'abord, nous envisageons le point de vue des processus et relations existants dans la conception comme dans la production des jeux. Nous discutons de comment les outils de conception peuvent être intégrés dans le flux du travail de conception (voir Figure 11), parallèlement à des processus de conception existants, ainsi que dans le processus de production des jeux dans son ensemble.

Nous discutons ensuite de l'interaction des outils de conception entre eux et développons les avantages et les inconvénients de l'utilisation d'un seul outil de conception de jeux. Enfin, nous examinons la relation entre outils et concepteur de jeux.

Nous nous appuyons sur deux observations : notre expérience durant la conception progressive de *The Casimir Effect*, et comment nous avons géré les besoins de la conception et de la production pour *The Particle Who Knew Too Much* en utilisant *Articy: Draft*.
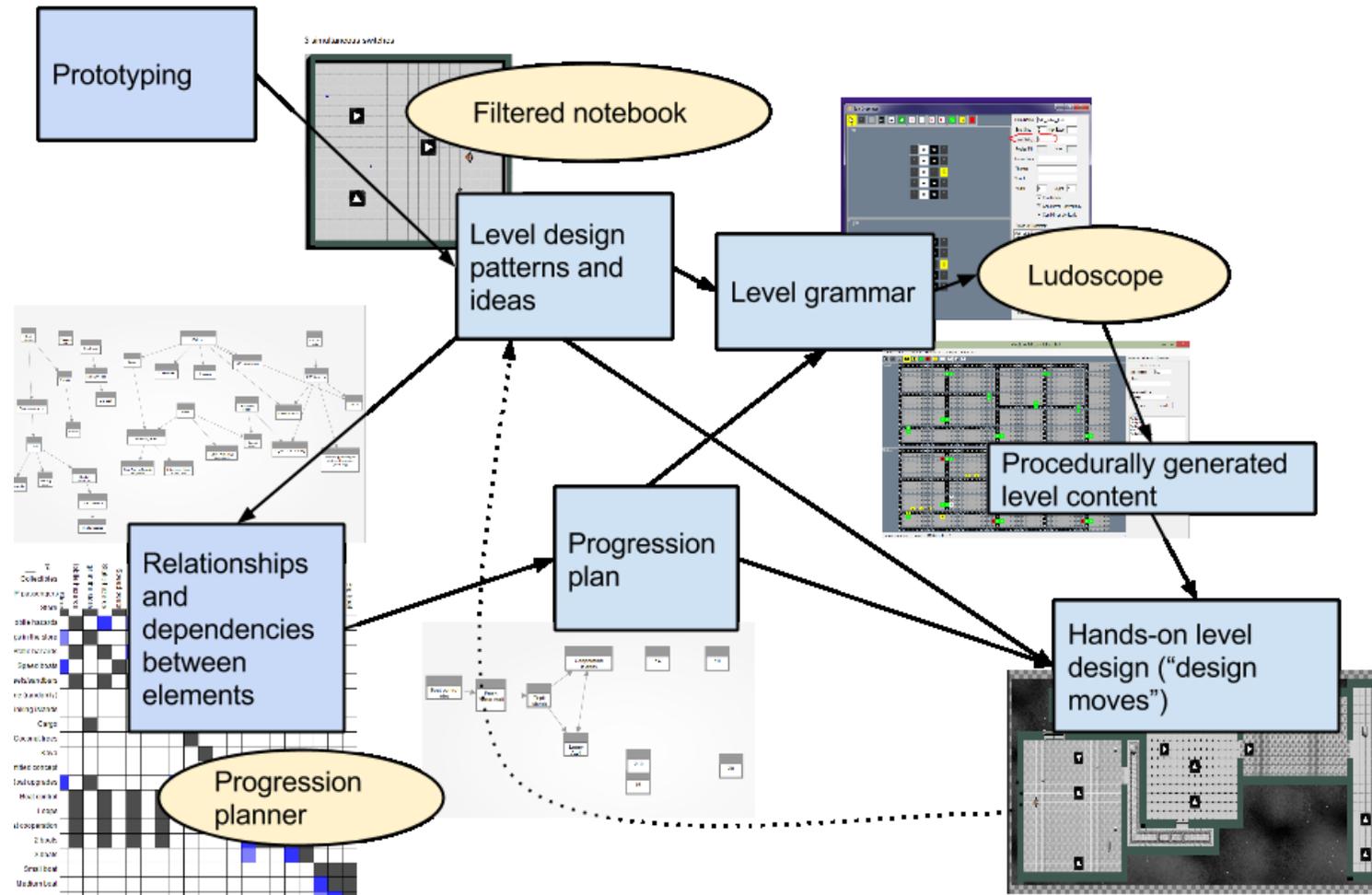
Figure 11 : flux du travail de conception de *The Casimir Effect*

Progressismo: Un outil de design de progression

Dans ce chapitre, nous expliquons comment nous avons construit un outil de design de progression basé sur l'approche de l'outil de «planification de la progression» de Butler (Butler et al. 2013). Notre objectif était de voir si cette approche pouvait être utile face à des problèmes de conception de progression plus exigeants et plus variés que ceux posés par *Refraction*, un jeu de puzzle linéaire. Notre méthode est d'adapter et d'étendre le concept de planification de la progression de Butler pour étayer la conception de progression pour nos études de cas.

Comme on le voit sur la Figure *,* le système construit par Butler comprend trois éléments qui permettent au concepteur de modifier par niveau leur design, partant d'un haut niveau d'abstraction jusqu'à un niveau très concret. Ceci prend en compte les contraintes, le plan de progression et les niveaux jouables eux-mêmes. L'article de Butler et al utilise le terme «contraintes» pour couvrir tous types d'éléments de design - toutes les règles régissant l'ordre des concepts de jeu - que le concepteur veut définir pour former son plan de progression.
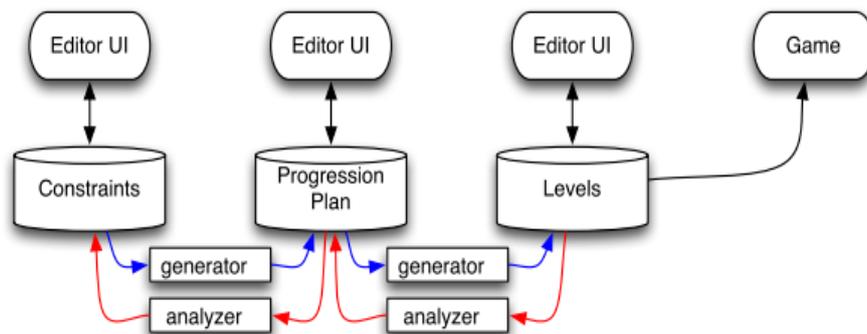

Figure 12 : modèle de système d'outils de *Refraction* (Butler et al. 2013)

278
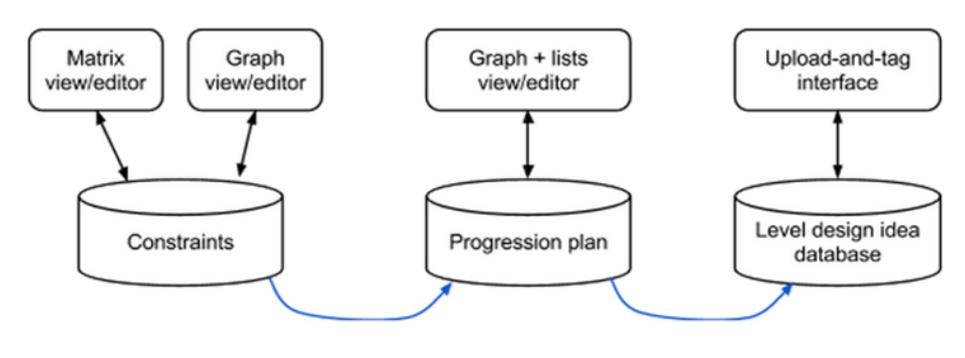
Figure 13 : modèle de système de *Progressimo*

Notre système *Progressimo* comporte les contraintes, le plan et le composant « référentiel des idées » (voir Figure). Comme l'outil de *Refraction*, il invite le concepteur à définir des éléments de progression (Butler et al. utilisent le terme « concepts de jeu ») et les relations entre eux (« règles de conception de progression », c'est-à-dire les contraintes) via un éditeur de matrice / grille. Le concepteur attribue des éléments de progression du plan pour chaque « moment » (pour *Refraction* ce sont des « étapes de jeu ») dans la composante de plan de progression de l'outil. Il est guidé dans cette tâche sur la base des règles de conception de progression. Ces listes d'éléments pourront ensuite être utilisées comme une sorte de « liste de courses » lorsque le concepteur va créer le niveau.

A la différence de l'outil décrit dans l'article de Butler, notre système, conçu pour satisfaire un large éventail de genres de jeux, n'inclue pas d'éditeur de niveau intégré ou de système procédural de génération de contenu (voir Figure).

Afin de prendre en compte les structures à monde ouvert et à chemins multiples de nos jeux, nous avons ajouté un éditeur et un analyseur graphique basés sur la structure de progression (voir Figure).

Figure 14 : une partie du plan de progression pour _South Sea Trouble_

Un autre ajout majeur est le composant Notebook (le composant « référentiel des idées » dans la Figure), qui fournit un moyen de stocker des concepts (notes) liés à chaque niveau. Les notes (ce sont des images) peuvent être marquées (*tag*) par les éléments de progression qu'elles référencent, ce qui permet à l'outil de les filtrer pour afficher un sous-ensemble de notes ne contenant que les éléments relatifs à la progression du niveau sélectionné.

Figure 15: le composant « référentiel des idées » de *Progressimo*

Progressimo a la possibilité de définir des portails (*gates*) de gameplay. Il s'agit des mécanismes donnant accès durant une étape du jeu à de nouveaux mécanismes. Il peut s'agir, par exemple, effectivement de l'accès à une nouvelle zone de l'espace virtuel ou, par la mesure de la progression des capacités du joueur, de lui permettre, lorsqu'un seuil est franchi, d'avoir accès à de nouvelles possibilités (armes, compétences...). Le *Gameplay gating* est un concept familier aux concepteurs de jeux. La définition et la gestion des portails de gameplay (basés sur l'XP ou la monnaie, par exemple) sont des tâches classiques de conception de la progression qui s'intègrent naturellement au concept de planification de la progression.

Enfin, *Progressimo* dispose d'un « mode d'exploration », avec lequel concepteur peut simuler l'achèvement de « tâche » prédéfinies (voir Figure 16). Dans ce contexte tâche a le même sens que dans un ordonnancement : une opération nommée qui a un début et une fin. Il peut s'agir, par exemple, d'un niveau de jeu ou d'une mission. Ici, nous avons introduit l'idée de « évaluation active », où le concepteur peut, assisté par l'outil, explorer la logique de son modèle de progression pour l'évaluer à son rythme. En mode « explorer » le concepteur simule la progression en cliquant sur des tâches accessibles (non bloqués), étape par étape, dans un ordre autorisé. A chaque étape où plusieurs tâches sont accessibles, le concepteur choisit laquelle jouer. L'accès aux tâches est contraint par des connexions et des portails. Le graphique est mis à jour à chaque étape pour montrer quels moments les tâches sont déverrouillées. Les problèmes de progression potentiels sont également mis en évidence.

Figure 16 : *Progressimo* **en mode « explorer »**

Conclusion

Sur la base de nos expériences, nous sommes arrivée à un certain nombre de conclusions sur la pratique de la conception de jeux avec des outils.

L'utilisation d'outils de conception de jeu peut être difficile. Nous soutenons que les concepteurs de jeux ont besoin de développer une meilleure compréhension formelle des systèmes de jeu afin de comprendre les représentations utilisées dans les outils. Les outils doivent eux même offrir des processus d'auto apprentissage à leurs formalismes. La simulation de la dynamique de jeu, par exemple, semble offrir un bon point d'entrée.

Notre travail de recherche nous a contraint à expliciter et formaliser notre pensée. Il a sensiblement impacté notre mode de raisonnement, non seulement pendant que nous utilisions des outils, mais a également impacté, en général, notre réflexion et les activités de conception. Ceci a fait évoluer la façon dont nous abordons la conception de jeux et constitue un cadre à l'approfondissement des concepts et de la pratique du Game Design.

En outre, l'utilisation des outils nous semble offrir de nouvelles possibilités de conception et de validation de ces possibilités. Nous avons constaté, en définissant nos propres règles de conception, que nous pouvons faire de la conception de manière plus sécurisée. Les outils permettent d'expérimenter une plus grande liberté, l'outil ayant en charge de nous rappeler les limites de cette exploration.. En particulier, la génération de contenu procédurale nous a permis de travailler simultanément à différents niveaux d'abstraction.

Les outils aident à l'organisation des moyens et des matériaux. La représentation, la structuration de ces matériaux permet, par exemple, de les produire de façon décontextuée. Peut-être que l'une des aides les plus puissants mise en évidence sont les outils de visualisation Les outils aident à structurer le processus et fournir une méthode de travail (*workflow*) pour aider à la conception des concepts le long d'un chemin vers le *gameplay* final concrétisé.

Dans ce contexte de structure et de sécurité, le soutien d'outils logiciels a eu pour effet d'élargir la portée des résultats de conception de jeu, permettant une plus grande complexité dans notre logique de progression et dans les systèmes de jeu.

Toutefois les outils exigent des efforts supplémentaires de la part du Game Designer. Il s'agit non seulement l'effort requis par les outils en termes d'apprentissage et d'utilisation, mais aussi les travaux préparatoires sur les matériaux de conception pour qu'ils puissent être modélisés et manipulés dans les outils.

Donc les outils nécessitent que le concepteur de prenne un point de vue réflexif non seulement par rapport à son activité de conception, mais également dans son utilisation des outils. La formalisation et la simulation ne doivent pas devenir des activités en soi mais un moyen de produire des contenus nouveaux et de meilleure qualité.

Nous avons abordé certains aspects de la façon dont les outils de conception peuvent être intégrés dans les méthodes de travail de conception et de développement existants. Nous avons noté la diversité des tâches dans lesquels ces outils peuvent être impliqués et la diversité des contenus produits, qui font sensiblement varier le processus de production. En conséquence aucun outil ne peut répondre seul à tous les besoins de conception et c'est donc une boîte à outil qu'il faut offrir au concepteur.

Nous avons constaté que les outils de conception peuvent jouer un rôle dans le prototypage de production, qui sont actuellement dévolus aux outils de développement. En outre, les outils de prototypage et de conception peuvent se compléter mutuellement. Par exemple les traces d'une simulation de conception peuvent servir comme base de scénarios de test et les comportements observés sur les tests d'un prototype peuvent alimenter la modélisation de conception.

Il est utile pour un outil pour permettre au concepteur de participer aux processus de conception à la « vitesse humaine » - nous avons constaté que lorsqu'un outil fait plus «intelligent» que le concepteur, il peut, dans certains contextes, être contre-productif. Nous avons proposé qu'un outil de conception concentre sur l'évaluation et l'analyse et laisse les tâches de validation et de type de vérification à d'autres types d'outils.

En construisant notre propre outil (*Progressimo*) nous avons constaté, en utilisant nos études de cas, qu'il est possible d'appliquer et d'adapter l'approche de planification de la progression de Butler à d'autres genres de jeux. Nous avons développé quelques idées pour étendre l'approche que nous avons intégrée dans notre outil.

En tant que chercheur, il y a nécessairement eu des limites à ce que nous avons été en mesure de réaliser dans ce projet. Le temps passé sur les tâches de production de jeu l'a été au détriment de l'activité de conception de jeu. Nous n'avons pas pu explorer autant que nous le désirions les tâches qui se posent vers la fin du cycle de développement, comme l'équilibrage du jeu. Idéalement, nous aurions entrepris cette recherche en tant que designer dans une équipe de production. Cela aurait réduit la charge en termes de développement et aurait également été plus représentatif d'une expérience industrielle.

Orientations futures de recherche

L'une des conclusions de nos travaux porte sur la relation des outils au type de jeux développés. Par exemple *Progressimo* a pu être utilisé au-delà de son contexte de création initiale en mettant en évidence des besoins génériques et en abandonnant les aspects trop contextuels. A l'inverse, lors de notre tentative d'utilisation de - *The Sentient Sketchbook*, outil qui nous semble à priori très utile, il a été mis en évidence un décalage entre les caractéristiques spécifiques de l'outil et l'étude de cas que nous avions l'intention d'utiliser. Le travail d'évaluation de l'aspect contextuel doit donc être prolongé dans le cadre d'autres jeux et d'autres environnements de création. De façon générale le processus d'évaluation par cas d'études doit être expérimenté auprès d'autres Game Designer. Ceci porte en particulier sur l'outil Progressimo que nous avons développé dans le cadre de cette thèse.

Dans le cadre de ce projet, nous nous sommes intéressée à l'idée d'explorer les relations possibles entre les outils et les méthodes qui reposent sur la production (par exemple prototypage et des mesures de joueur) et des outils de conception purs. Il existe une gamme de données actuellement utilisée dans le cadre des outils d'analyse, au service de la conception axée sur les données. Une question de recherche future est de savoir si ces données pourraient être utilisées dans le cadre des outils de conception de jeu, au service d'une conception réflexive.

Enfin de nombreux concepteurs de jeux se sentent menacés par l'utilisation de la technologie dans la conception, pouvant être utilisée contre eux, ou même à terme les remplacer. La manière dont une pratique de conception soutenue par des outils peut être adopté par la communauté du design de jeu (et de là, donner lieu à un discours de praticien) demeure un défi important et digne d'un travail de recherche en soi.

	Katharine NEIL	
le cnam	Game design tools : Can they improve game design practice ?	Flinders UNIVERSITY

Résumé :

Cette thèse constitue une recherche expérimentale pour évaluer la capacité des outils d'aide à la conception du « Game Design » de jeux vidéo. Elle cherche à dégager les éléments destinés à instrumenter et à améliorer le développement de ces outils.

La production des jeux vidéos s'appuie sur de nombreux outils techniques allant des éditeurs graphiques et sonores jusqu'au moteurs de jeux. Mais, en amont, et à l'inverse des praticiens d'autres domaines du design, les concepteurs de jeux (Game designers) utilisent généralement peu d'outils dans leur travail quotidien. Bien que des modèles conceptuels et des outils logiciels aient été développés pour les assister, il manque une étude approfondie sur la pratique de ces outils. En prenant le point de vue d'un « Game Designer », notre étude se base sur une étude de terrain, menée par un participant–observateur, utilisant les outils sur 5 études de cas.

Nous analysons dans un premier temps les pratiques du Game Design : les différentes activités impliquées et leur mise en œuvre dans le processus de conception. Nous nous intéressons plus particulièrement aux développements et tendances actuelles.

Le chapitre 3 positionne le Game Design dans le contexte plus général des travaux en cours sur la critique et l'analyse du Design (Design Studies). Nous en concluons que le Game Design se situe plus comme dans le domaine de la conception artisanale (« craft-based ») et que l'utilisation d'outils serait une évolution vers une conception réflexive (« self-conscious »). Cette évolution constitue donc, à notre avis, une rupture par rapport aux pratiques actuelles.

Nous retraçons l'histoire du développement des outils de Game Design tant du point de vue de la recherche que de l'industrie. Ceci nous permet, au chapitre 4, de donner une définition plus précise et comparative des outils de Game Design.

Le chapitre 5 formalise les questions de recherche de cette thèse et la nécessité d'une évaluation basée sur la pratique. Le point de vue méthodologique de notre travail est ensuite développé.

Nous présentons ensuite nos observations et analyses d'expériences réalisées en utilisant une sélection d'outils reposant sur différentes approches. Parmi ces outils citons: « Articy:Draft », « Ludoscope » et « Progression Planning ». Nos critères d'évaluations sont tirés des publications sur le « Design Support » et le « Creativity Support Tool ».

Au chapitre 10 nous décrivons et discutons les expériences effectuées avec un outil de design que nous avons développé dans le but d'évaluer et d'étendre l'approche « progression planning ». Nous explicitons les choix de conception de l'outil issus en partie du travail d'analyse mené dans cette thèse.

La conclusion résume nos observations et introduit des principes d'amélioration des outils de « Game Design » et propose d'étendre nos travaux sur différents types d'outils et contextes d'utilisation non abordés ici.

Mots clés :

jeu vidéo, game design, conception, outil de game design, étude de terrain

	Katharine NEIL Game design tools: Can they improve game design practice?	

Abstract:

This thesis contributes practice-led evaluation research to the question of whether game design tools can effectively support and expand game design practice. It offers insights that can be used to inform future game design tool development.

Game designers, unlike most other design practitioners, typically do not use design tools in their work. While conceptual models and software tools have been developed to address this, we lack discussion and critique of how tools work in practice.

Taking the point of view of a practitioner, this study of game design tools is based on longitudinal, practice-led evaluation research conducted as a participant-observer, applying game design tools to 5 contrasting game design case studies.

Design tools support game design practice. In Chapter 2 I set out what that practice is today, reviewing the game design process and design activities, and giving particular attention to contemporary trends and directions**.**

Chapter 3 situates game design within the broader context of Design Studies. After reviewing relevant ideas from this literature, I conclude that game design is best characterised as a crafts-based design discipline, and that adoption of design support would mean a shift towards "self-conscious" design. I argue that, within this context, the adoption of tools into current practice would represent a significant and disruptive change to current practice. I then trace the history of development of game design tools within the game design and research communities.

Chapter 4 offers a more precise definition of game design tools and presents a comparative review of the tools within this scope.

In Chapter 5 I argue for the need to bring practice-led evaluation research to this question. Here I set out my research goals and questions, before specifying the particular tools I practiced with for this study. I discuss methods used in relevant research fields and outlines the method used for this project.

Chapter 6 introduces the game projects I used as case studies and describes how I worked with each. Chapters 7 and 8 present observations and analyses of my experiences using a selection of tools that vary in their approaches. These include (though not exclusively): *Articy:Draft* (Nevigo GMBH 2011), *Machinations* (Dormans 2009), *Ludoscope* (Dormans and Leijnen 2013), and "progression planning" (Butler et al. 2013). Evaluation themes and criteria are drawn from the Design Support and Creativity Support Tool literature. Chapter 9 refocuses attention on the practicalities, addressing the question of how game design tools integrate into the wider context of practice.

In Chapter 10 I present and discuss my experiences with my own game design tool, which I prototyped in order to evaluate and extend one of the tool approaches under study. I explain my tool design choices, some of which reflect the knowledge I have acquired through the course of this study.

Chapter 11 summarises my conclusions in relation to how design tools might best support game designers, and offers ideas for further practice-led evaluation research related to this question.

Keywords:

game design, design tools, practice-led, videogames, evaluation research