Orion, A Generic Model for Data Mining: Application to Video Games

Julien Soler

université de bretagne occidentale

Thèse / Université de Bretagne occidentale
sous le sceau de l'Université européenne de Bretagne

pour obtenir le titre de
Docteur de l'Université de Bretagne occidentale
Mention : Informatique
École doctorale Sicma

Orion, A Generic Model for Data Mining: Application to Video Games

présentée par
Julien SOLER
préparée au Centre Européen de Réalité Virtuelle, ENIB
Laboratoire Lab-STICC

Thèse soutenue le 08 septembre 2015
devant le jury composé de :

Fred Charles (rapporteur)
Principal Lecturer, Teesside University, Royaume-Uni

Patrick Reignier (rapporteur)
Professeur des Universités, ENSIMAG, France

Humbert Fiorino (examinateur)
Maître de Conférence, Université de Grenoble 1, France

Antonio Mora (examinateur)
Chercheur, Universidad de Granada, Espagne

Vincent Rodin (examinateur)
Professeur des Universités, UBO, France

Cédric Buche (directeur)
Maître de Conférence HDR, ENIB, France

Laurent Gaubert (invité)
Maître de Conférence, ENIB, France

Fabien Tencé (invité)
Ingénieur de recherche, PhD, Virtualys, France

Abstract

This thesis takes place within the context of designing a behavior model for controlling entities in video games.

In chapter 1, we propose to identify the current and future needs of Artificial Intelligence in video games by means of a historical review of video games. We will then discuss the purpose of AI. How is artificial intelligence used? What features enable it to be implemented? What additional constraints are necessary? With this review we will be able to show how imitating real players seems to be a promising and fruitful strategy.

In chapter 2, we show that in order to consider the development of a generic software solution for designing artificial intelligence in video games based on replicating human behavior, it is pertinent to "ask the question" of the requirements of such a system. So we start by identifying the different requirements related to constraints in the development of a video game. Secondly, we focus on the characteristics of human behavior in the context of a video game. These features define requirements on the expressiveness of the models to carry out such behavior. And since we want to learn these behaviors, we are also interested in learning requirements by themselves. These different criteria allow us to put into perspective the relevance of different models in the context of the construction of the proposed model. Since we want to reproduce the behavior of a human by imitation, we believe that the solution must be oriented by data retrieved from real humans operating in the environment of the game. Data mining techniques offer an interesting range of techniques that can be useful in the context of this approach. We present a state of the art of data mining techniques and we evaluate them based on the requirements defined above. We conclude that all of these techniques can be useful in developing our solution and the solution must allow the use of several of them. However, they alone are not sufficient to fulfill all the criteria we have identified.

In chapter 3, we present our contribution: the ORION model. This model is composed of a structural part and a behavioral part. The structural model allows datasets to be handled generically and semantics to be added to data.

The behavioral part is based on behavior trees. They meet many of the needs expressed previously. We propose an extension which makes it possible to construct an artificial intelligence from data retrieved from real players. When using behavior trees, the data is usually stored in a key-value dictionary used by all behaviors. The structural model helps to streamline inputs and outputs used by the model. ORION offers data typing, allowing the semantics of data to be maintained throughout the design, as well as checking the consistency of the data used by each behavior. We show, thanks to our implementation of the model, the flexibility of our approach by manipulating and visualizing various data sets. The behavioral model, although based on behavior trees, enables behaviors performed by machine to learn both online (during the execution of the game) or offline (during the design phase). These behaviors can produce additional knowledge from data (such as automatic learning of the environment) or automatically partition the set of sequences in order to identify different strategies used by the player.

In chapter 4, we finally show how the proposed model can be used on concrete examples. This model can be used in many different ways, using expert added knowledge or not, performed online or not. First, we use the model in order to implement a behavior able to play the video game *Pong*. Then, as Tencé (2011) proposed a technique to automatically learn the environment in the game *Unreal Tournament*, we propose to enhance these techniques using the proposed model. We then, implement a bot able to perform better than Tencé's agent model. As the implemented bot still lacks many important features, we conclude by proposing a potential solution to fix issues that hinder our model.

Contents

List of Figures

List of Tables

List of Acronyms

AI Artificial Intelligence

BT Behavior Tree

BW Baum-Welch algorithm

DBN Dynamic Bayesian Network

DTW Dynamic Time Warping

EM Expectation-Maximization algorithm

FFNN Feed Forward Neural Network

FPS First Person Shooter

FSM Finite State Machine

GNG Growing Neural Gas

HMM Hidden Markov Model

IOHMM Input-Output Hidden Markov Model

KNN K-Nearest Neighbors

MLP Multi-Layer Perceptron

MSE Mean Squared Error

NPC Non Player Character

PCA Principal Component Analysis

SGNG Stable Growing Neural Gas

SOM Self-Organizing Map

SVM Support Vector Machine

UT3 Unreal Tournament 3

Contents of Chapter 1

Chapter 1

Introduction

The past is just a story we tell ourselves.

Samantha - Her (2014)

Summary

In this chapter we propose to identify the current and future needs of Artificial Intelligence in video games by means of a historical review of video games. We will then discuss the purpose of AI. How is artificial intelligence used? What features does it enable to implement? What additional constraints are necessary? With this review we will be able to show how imitating real players seems to be a promising and fruitful strategy.

Introduction

Despite relatively recent development, in most technological societies, there is nowadays a real infatuation around video games, with an increasingly wide audience. Video games inspire other forms of art and entertainment (such as cinema and literature), to the extent that there has been much debate around the concept of a new kind of art (Smuts, 2005).

Introduction

Video games are also a source of interest to the scientific community. The number of publications in this field is growing exponentially. They are a source of interest for scientists from varied fields such as sociology, psychology (ergonomic and cognitive), pedagogy, and of course computer science. As video games can be very complex environments, they are well-suited to implementing and evaluating many types of algorithms, such as learning algorithms.

However, the reverse is less true: the video game industry does not make full use of this scientific involvement and uses this resource marginally in order to develop artificial intelligence. For instance, the techniques used to govern the behavior of Non Player Characters (NPCs) are quite limited: simple scripts, finite state machines, etc. Moreover, in most games, artificial intelligence is primarily intended to increase the performance of NPCs. While this approach can offer players a real challenge in order to defeat the computer (in the case of NPC opponents), it can also be frustrating for the human players. This is because excessive performance can even lead to a loss of interest. On the other side, believability is at least as important as performance. Human-like NPCs really can help the player's immersion, as he can use his knowledge on humans to reason about the NPCs so as to complete a task. Therefore, the real difficulty that developers face is the design of NPC's artificial intelligence that achieve a balance between the challenge proposed to the players and the credibility of their artificial entities. We consider that a natural way of reproducing this behavior is to observe how real players operate in a game. But machine learning algorithms are usually quite complex and have a high computational cost, this is why game development studios are reluctant to use this strategy. It is, therefore, a real challenge to design a suitable solution to imitation learning of behaviors which is both efficient and robust.

Plan

In this chapter we look back at the history of video games and focus especially on artificial intelligence. As the use of AI in this medium is becoming increasingly common, we propose to identify the reasons and goals for incorporating it. This will lead us to precisely identify the current and future needs of this vast industry. Then, we will discuss how learning by imitation may fulfill many of these needs.

1.1 History of Artificial Intelligence in Video Games

1.1.1 Prehistoric Age

The video game history is quite complicated to trace back. Very early video displays like Braun's Cathode Ray Tube (CRT) in 1897 may have been used by researchers to experiment with entertaining activities. To our knowledge, the very first device that can be considered a video game is Thomas Goldsmith Jr.'s cathode ray tube amusement device patented in 1947 (Goldsmith, 1948). It consists of an analog circuit controlling a dot on a CRT screen simulating a missile trajectory.

In 1951, the NIMROD computer was presented during the Festival of Britain. This computer was specially designed to play the Nim game with a human. One year later, in 1952 during his PhD, Sandy Douglas developed one of the very first games running on a general purpose computer: *OXO*, a tic-tac-toe game running on the EDSAC computer. It was developed in order to create a human-computer interface built with a rotary dial. Each box of the tic-tac-toe grid was attributed a number and the user was able to play by means of dialing the corresponding number. The opponent was the machine. The algorithm defining the opponent behavior was designed to be optimal so that the player could not win. This was probably one of the first AI used in a video games. Those two games are perfect examples of **problem solving** abilities of an AI in video games.

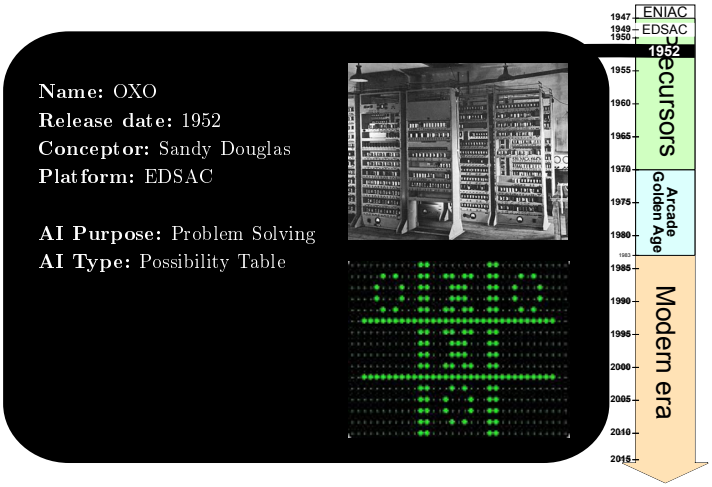


Figure 1.1: EDSAC

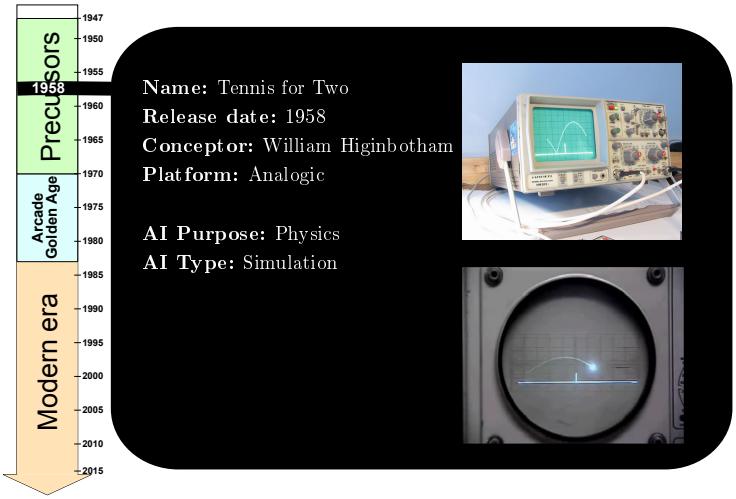


Figure 1.2: Tennis for Two

A few years later, more accessible hardware was available in universities and was powerful enough to create video games. As a result, in 1958, William Higinbotham designed an analog computer game simulating the physics of a tennis ball: *Tennis For 2*. The graphical display was an oscilloscope and players had to use home made controllers consisting of a potentiometer and a button. This did not implement any AI as both opponents were human. Nevertheless the physics was implemented providing the ball with a quite realistic behavior. This is an illustration of the early need of **autonomous behaviors** for interactive objects in video games. Another good example of early physics implementation is *Spacewars!*, developed in 1961 by MIT students and running on PDP-1 computer (one of the first micro computers). This game is a spaceship battle simulation. The two opponents can launch missiles and control the orientation and ignition of a rocket engine. A black hole located in the middle of the screen attracts the spaceships. While trying to destroy each other, the two players have to control their trajectory using engine ignition in order to avoid being trapped inside the black hole.

At the beginning of the 70's, video games gave rise to an actual industry. They were placed next to pinballs and other arcade games. Many companies, like Atari, Taito, Nintendo or SEGA, were either founded for, or focused their activities on developing video games. Games like Atari's *Pong* became one of the first video games to reach the status of popular success. But the golden age of arcade video games really took place at the end of this decade.

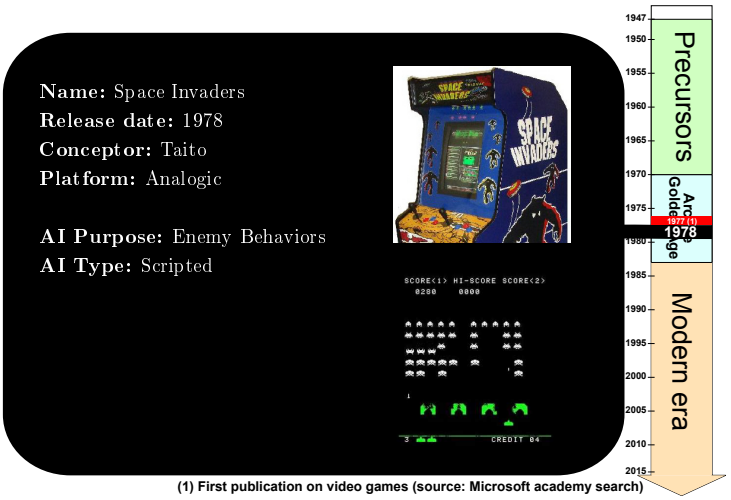


Figure 1.3: Space Invaders

Released in 1978 by Taito, *Space Invaders* is a "shoot 'em up" game considered in popular culture as one of the most famous games in history. The player controls a spaceship horizontally at the bottom of the screen and must shoot at alien waves with a raygun. The behavior of aliens is totally predictable as they just wave from left to right and move down. This is an example of **scripted autonomous behavior**. This allows the player to easily elaborate a strategy to destroy all aliens before they reach the ground. Despite being basic, this kind of "Intelligence" was sufficient to provide a real challenge as it was coupled with an increase in the aliens' speed.

Figure 1.4: Pacman

Pac-Man (Namco, 1980) is another masterpiece of arcade video games. The player controls Pac-Man, a round yellow character, opening and closing his mouth as he travels. He operates in a maze where ghosts are chasing him. The player has to collect dots scattered in the level while avoiding the ghosts. However, some dots (power pellets) make the ghosts vulnerable for a limited time. Particular attention was paid to the behavior of these ghosts. Each has its own personality. Thus, Blinky (red) continuously chases Pac-Man while Pinky (pink) and Iky (blue) simply try to get in front of him. Finally, Inky (orange) has a somewhat random behavior. The design of these behaviors is the result of reflection by Toru Iwatani, designer of the game, who wanted to find a compromise between the difficulty of the game and the player's interest. In this context, we observe that Artificial Intelligence (AI) naturally became a crucial part of the game. It was a tool that helped designing what will be later on named **the flow**(Csikszentmihalyi, 1991): the abstract area where the player feels a deep enjoyment of a certain

activity.

1.1.2 Modern Era

In 1972, the Magnavox Odyssey marked the beginning of the game console market. These consoles used logic circuits which directly implemented the games. Therefore the games that they would run had to be planned from the outset of their design. Later on, in the late 70's we saw an explosion of this market, due to the arrival of the second generation of game consoles. They integrated microprocessors, like for instance the legendary Atari 2600. This second generation of consoles surpassed the limitation of embedded games. As a consequence, during this period and in the early 80's, the market was flooded with games that were quite similar (clones of *Pong*, *Pac-Man*, *Space Invaders* or *Arcanoid*) and sometimes poorly developed. This situation led to the video game crash of 1983. This crisis ended with two major steps from Nintendo: on the one hand it released the game console named NES (Nintendo Entertainment System), and on the other hand it imposed a new economic model that made the game industry sustainable. This period saw the appearance of franchises which are still widely popular today: *Mario*, *Zelda*, *Bomberman* for Nintendo and *Sonic* for Sega. This was what we now call the modern era of gaming. However, there were no major advances in the use of AI for video games during this era.

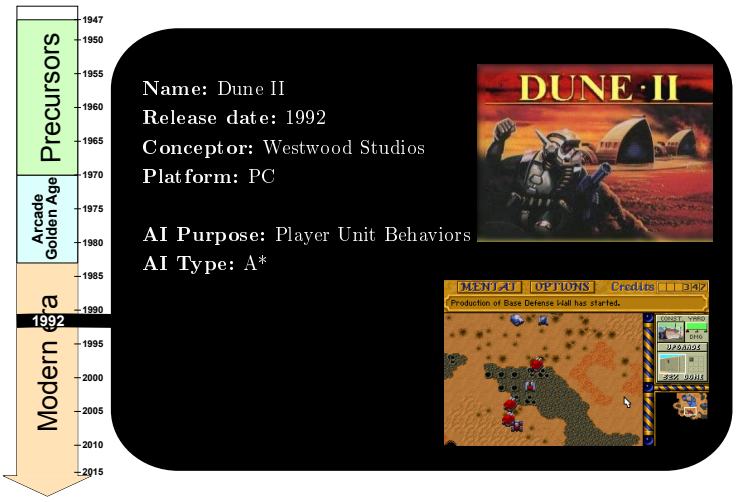


Figure 1.5: Dune II

Based on Frank Herbert's famous science-fiction novel, *Dune II* (Westwood Studios, 1992) is a game which deserve mentioning. Although it did not revolutionize AI in games, it laid the foundations for a new kind of video games. In this real time strategy (RTS) game, the player exploits resources in order to build a colony (mainly buildings and combat units) and to fight enemy armies. The AI of the units is relatively simple but provides an essential function to control these units effectively: **path finding**. This way, the player simply selects units and gives high level orders: "move to a given location" or "attack an enemy unit". Therefore, the units in question have to follow a path, avoiding potential obstacles. While it is difficult to know which algorithm was used in *Dune II*, nowadays, path finding is usually done

using an implementation of the A* algorithm (Hart et al., 1968). This algorithm is one of the most commonly used in video games to solve some **optimal solution problems** (path finding problem being just one application of it). This game mechanism was subsequently widely used by successive titles like *Warcarft* and *Starcraft* or the games of the *Command and Conquer* franchise.

Worms, published in 1995 by Team17, showed another use of AI: **automatic content generation**. Worms is a turn based artillery game. Players control the worms of their team alternately and must choose between various weapons in order to eliminate the opposing teams. The land used as a 2D fighting arena is randomly generated while remaining consistent and playable. This gives the game an uncommon lifespan, since billions of game levels are theoretically available.

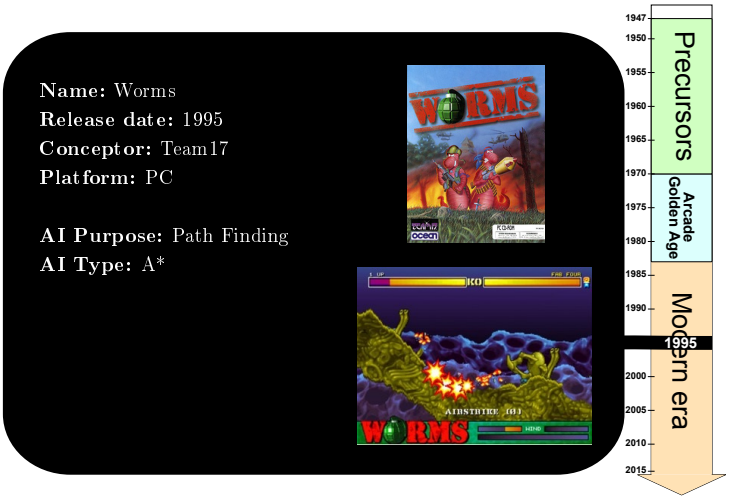


Figure 1.6: Worms

Now, if we were to find, in the entire history of video games, one game that stands out particularly for its AI features, it would definitely be *Creatures* (Millenium, 1996). The goal of the game is to raise an artificial life. In line with Tamagotchi, released the same year, the player must take care of virtual creatures called Norns. These creatures can play, learn, breed, feed and protect themselves from other evil creatures (the Grendels). In this game, **learning** is a gameplay feature. Relatively advanced AI techniques are used such as **genetic algorithms** and **neural networks** (Grand and Cliff, 1998).

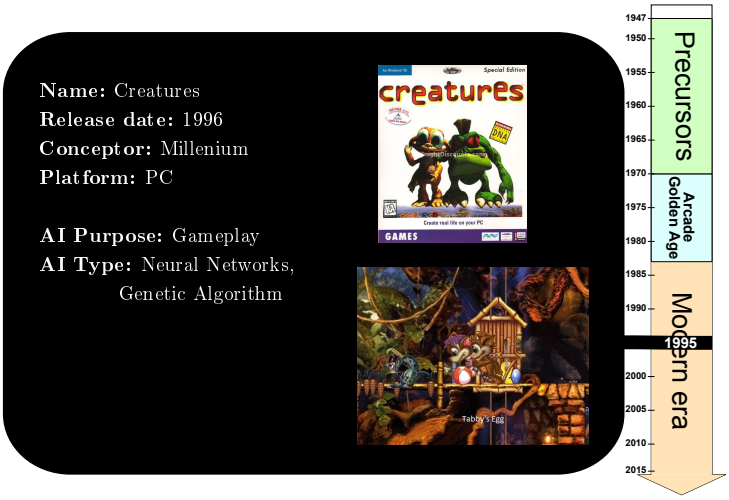


Figure 1.7: Creatures

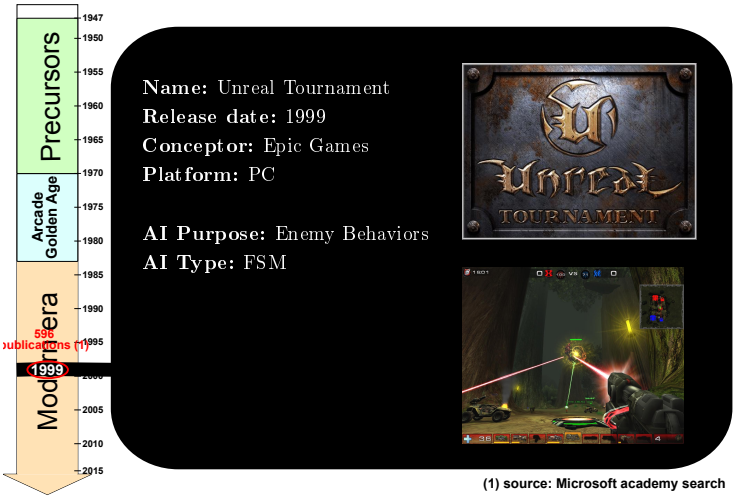


Figure 1.8: Unreal Tournament

Three years later *Unreal Tournament* (Epic Games, 1999), a First Person Shooter (FPS), was released. In this game, the behavior of non-player characters (enemies or teammates), was implemented using a Finite State Machine (FSM). Of course, Unreal Tournament was not the first game to use this technique, but the mechanism was incorporated directly into the game engine (Unreal Tournament was primarily designed to showcase the Unreal Engine). This allowed developers to create other games using the UnrealScript language which includes the mechanism of FSM. The convenience was to facilitate and thus accelerate the development process, inciting code **re-usability** and **robustness**. On top of that, the behavior thus developed was deterministic and **testable**. Unreal Tournament led to some simplifications in the development of video games in general, and in AI of NPCs in particular.

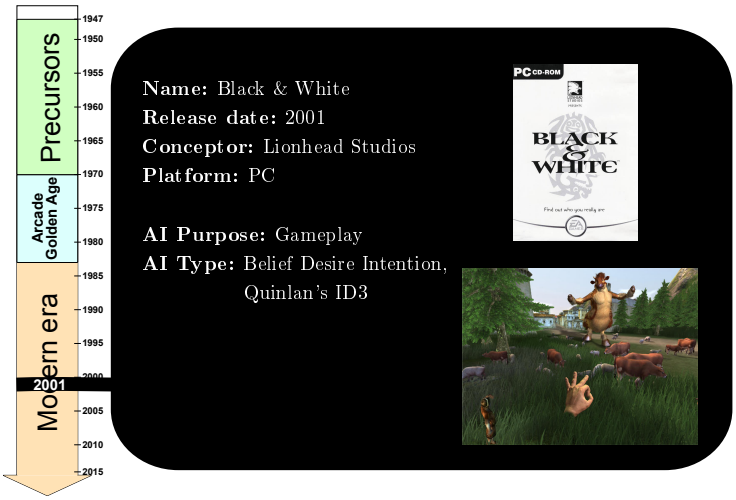


Figure 1.9: Black & White

Black & White (Lionhead Studios, 2001), where the player embodies a god, uses the same approach as *Creatures*. The player has to teach his servant creature to help him reign over his people. While, once more, learning is the main element of the gameplay, the techniques are not the same. In contrast to genetic and connectionist methods, the player's servant creature is implemented with the **belief desire intention** (BDI) architecture and a **decision tree**, while learning is achieved through Quinlann's ID3 (Quinlan, 1986) (Evans, 2001).

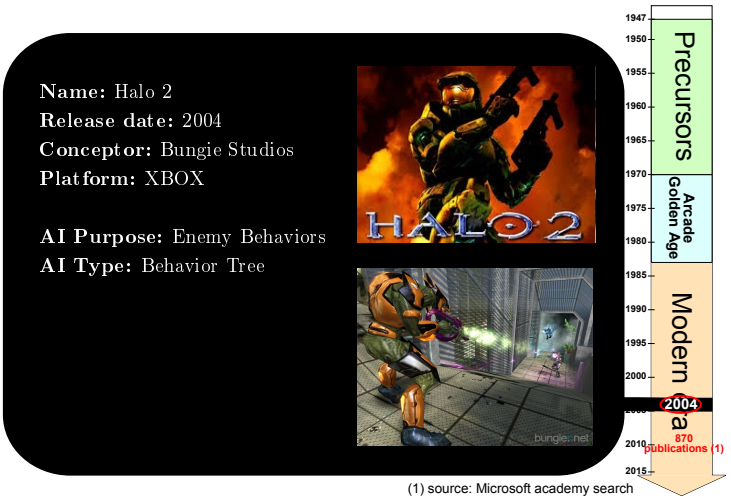


Figure 1.10: Halo 2

However, FSMs quickly showed their limits when developers had to address complex cases. Quite often, the behaviors they had to design depended on many parameters such as the level's difficulty or the game type (deathmatch, team death-match, capture the flag, and so on). At this point, FSM became extremely complex and less **maintainable**. The FPS *Halo 2* (Bungie Studios, 2004) is probably the most relevant example of the use of a new approach in order to overcome this pitfall: **Behavior Trees (BTs)** (Isla, 2005). A behavior tree represents the behavior of the agent by means of a tree structure. Leaf nodes represent actual behaviors while non leaf nodes are used to make choice between their child nodes. Several types of non-leaf nodes exist, resulting in a high level of expressivity of the model. Furthermore, a sub-tree may be referenced in several places within the main tree. The main benefits of this representation model are that it is highly human readable, debugging tools can be easily implemented and entire subparts of a behavior can be easily reused.

Figure 1.11: F.E.A.R.

F.E.A.R. (Monolith Productions, 2005) used a different approach. In order to simplify the development and to get more believable behaviors, the game exploits **a planner** (even if, for technical reasons, the implementation uses a FSM). Several basic actions are implemented such as reload, take cover or suppression fire, and are associated with preconditions and effects on the state of the world. Each agent has a number of goals that are prioritized. The planning algorithm (here, A*) generates an action sequence that will lead to the achievement of these goals (Orkin, 2006). The main benefit of this approach is the simplicity of adding new features. As an example, suppose you need the NPCs to turn on the lights when entering

a dark room. With a FSM, developers need to add this feature in several states (combat situation, patrolling, ...). With the planner, he just needs to add a precondition "lights should be on" on the action handling movement and to implement an action that changes the state of the world to "lights on". Moreover, each action is associated with a cost, and the planner provides the solution with lowest total cost. When an action fails, it is possible to fall back to another solution of higher cost, giving the illusion that the NPC can change its "mind".

Figure 1.12: Left 4 Dead

In recent years, many games proposed ambitious AI systems. In the game *Left 4 Dead* (Valve Corporation, 2008), the AI system, called Director AI, is responsible for placing objects and enemies in the level to encourage **replayability** (Booth, 2009). Moreover, this system can adapt the game difficulty according to the players' skills: this is named **dynamic difficulty adjustment**. It is also involved in the rhythm of the game, alternating stressful sequences and calmer ones, creating a real atmosphere specific to the game.

Figure 1.13: Forza Motorsport 5

The driving game *Forza Motorsport 5* (Turn 10 Studios, 2013) uses mechanisms that come from academic research in order to build what the developers named Drivatar. When a player drives his virtual car, the program records the driving session. Then, the software analyses these data and design the drivatar. When the player is offline, this drivatar will be able to reproduce his behavior so that other player may still face him off. The player is encouraged to

train his Drivatar through two-phased learning sessions. Initially, the player is asked to drive alone on the track. Opponents are added in the second phase in order to learn how the player behaves in traffic. **Learning is really a part of the gameplay** as the player attempts by multiplying the training sessions to make his Drivatar as accurate as possible. This is a rare example of the use of **imitation learning** in video games.

1.2 Imitation Learning: The Next Challenge of Video Games' AI?

This review allows us to identify the main uses of AI in video games as well as the constraints on the techniques to be used. As a summary of that review, we have seen that AI in video games aims to:

- Improve the performance of non-player characters (as opponents or friends)
- Enhance the believability of NPCs.
- Propose innovative gameplay.
- Promote replay value.
- Adjust the game's difficulty.
- Automatically generate content (maps, story arc, etc)
- Solve game-specific problems (pathfinding, choosing an optimal strategy, etc)

If the answers to game-specific problems and automatic content generation seem to be side issues, the other use cases may be addressed by a generic imitation learning solution. Indeed, with an effective solution, the performance of the reproduced behavior would be directly related to the level of the teacher player. In the case of online learning, it could be possible to propose gameplay based on this learning (as in *Creatures*, *Black & White* or *Forza Motors*). Moreover, online learning with the player as the teacher would cause a change in the NPCs' behavior. Therefore the player would still face a NPC with adapted skills. Still in the case of offline learning, real life data could be used. It would then be possible to learn the behavior of an entire football team or of a real tennis player for example.

The development of a generic software suite for learning behavior by imitation would address, to some extent, these questions. The development of an artificial behavior would then amount to a learning session. One may compare this mode of development to motion capture. This process has resulted in

significant time saving in the production of humanoid animations. Without motion capture, a graphic designer has to decompose all movements manually. Motion capture can significantly speed up the animation production process and allow even more believable animations. "Behavior capture" seems to be an interesting option for the future of AI in computer games. To the best of our knowledge, the result of current imitation learning techniques are far from convincing in all situations. The aim of this thesis is to propose ways to make a further step towards this challenge.

1.3 Objective of our Work

With this brief study of the history of AI in video games, we provided clues on future video game industry needs in AI. We then found that many of these needs could be met through an imitation learning system.

The main objective of this thesis is to propose a **generic model** for implementing AI in video games. This model aims to provide **imitation learning based on human game traces**. By generic, we mean that our solution should be able to be **applied to different video games**. To provide imitation learning, we are considering the **use of data mining to analyze human player behavior** and **coupling data mining techniques with bot behavioral model**.

1.4 Organization of this Manuscript

This manuscript is organized with the following plan.

In **chapter 2**, we will study how tasks learned by data mining techniques can be useful to implement a behavior imitation learning solution based on human player behaviors. Data mining includes many techniques such as data transformation, features extraction, vector quantization, data prediction, etc. All these techniques are very useful to imitation learning. Data mining can be used to extract knowledge on the environment or on other players, to reproduce purely reactive behaviors, to identify sequences of games where the player was performing similar tasks, etc. We will review a panel of families of techniques. We will see that each of them has its weaknesses and qualities.

In **chapter 3**, we will develop our model proposal. The proposed workflow of this generic architecture is presented figure 1.14. This model first proposed to model datasets by adding semantic. Using semantic allows, first, to facilitate their understanding. Thus, it is possible to use specific data representations adapted to the semantic. Secondly, some data transformations or calculations are only applicable to certain types of data (as an exemple, DeltaE distance is only applicable between 2 colors). Our model proposes a generic architecture

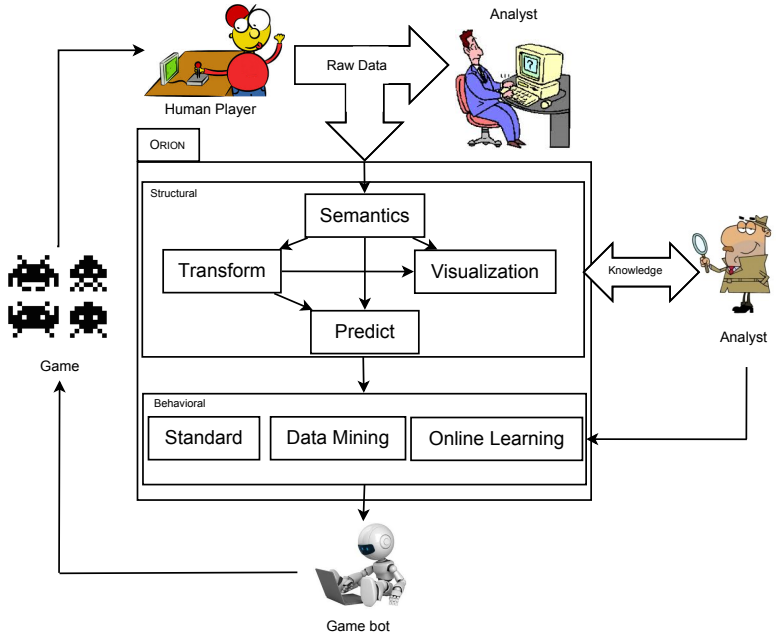


Figure 1.14: ORION Workflow

for data visualization, processing and prediction, either the main tasks achievable by data mining techniques. Finally, our model offers a behavioral part, flexible enough to implement any behavior since it is an extension of a model used in video games. We extended that model in order to be able to perform, full or partial, online or offline, learning.

In **chapter 4**, we will use our model to implement the behavior of an agent for *Pong* video game. This learning is made online, using several data mining algorithm and reproduces satisfactorily the behavior of a human playing to *Pong*. We will also use our model to implement an agent for the game Unreal Tournament 3 (UT3) . Learning here is more partial and is performed offline. Although the resulting behavior is not yet able to pass a behavioral evaluation, as a Turing test.

In **chapter 5**, we will conduct a review of the work presented in this thesis. We will summarize the contributions of our work and expose those which, in our opinion, are the most significant. We will expose the weaknesses of our model and will try to define areas of research to overcome them.

Contents of Chapter 2

Chapter 2

AI Solutions for Video Games

> I've seen things you people wouldn't believe. Attack ships on fire off the shoulder of Orion. I watched C-beams glitter in the dark near the Tannhäuser gate.
>
> Roy Batty - Blade Runner (1982)

Summary

In this chapter, we study the development of a generic software solution for artificial intelligence design in video games. We first list the requirements such a system would need. The first requirements are classic constraints in video game development. Next, we focus on the characteristics of human behavior in a video game context. These features help us to define requirements on the expressiveness of the models. Since we want to learn these behaviors, we are also interested in the requirements of the learning process itself. These different requirements allow us to put into perspective the relevance of different models in our context.

We want to imitate human behavior, therefore we need to operate with data from real humans operating in the game environment. The field of Data Mining offers an interesting range of techniques that can be useful. Weber and Mateas (2009) has already successfully used some data mining techniques to implement a real time strategy game bot. Thus we evaluate various data mining techniques, according to the requirements defined previously. We conclude that all of these techniques may be relevant and that our solution must allow the use of several of them. However, these techniques are far from being sufficient to fulfill all our requirements.

Introduction

The development of a generic solution for artificial intelligence design in modern video games presents a great challenge. We argue that current systems and attempts can be improved especially by means of imitation strategies.

Plan

In section 2.1, we consider the requirements of such solutions according to three different points of view: development, expressiveness and learning.

In section 2.2, we discuss the concept of data-oriented design. We believe that, since we want to reproduce human behavior, it is pertinent to consider players' data as the cornerstone of our solution. We explain how data mining techniques can be effective in order to extract knowledge from these data and how they make behaviour reproduction possible. We then study various data mining tasks and evaluate them from the perspective of the requirements identified in section 2.1.

2.1 A Modern AI System for Video Games: Requirements

The design of a generic Artificial Intelligence (AI) solution for video games needs to take many requirements into account. First (section 2.1.1), we list the development constraints and identify a paradigm that would suit our needs.

Then (section 2.1.2), as we want to develop a learning system, we define criteria related to the characteristics of the behaviors we want to reproduce. Finally (section 2.1.3), we consider learning problems. These two types of requirements will allow us to identify the strengths and weaknesses of the different methods we may use.

2.1.1 Development Requirements

As in any complex project, the development of a video game involves many people over a long period of time. Many constraints appear relating to the general architecture of the solution. The associated requirements that we have identified are listed in table 2.1.

Development requirements	Summary
[D1: Maintainability]	It should be easy to fix or improve
[D2: Scalability]	The solution should be scalable
[D3: Flexibility]	It should be possible to add *ad hoc* features
[D4: Testability]	The solution must be testable
[D5: Communication Support]	The solution should facilitate communication between teams

Table 2.1: *List of Development Requirements*

[D1: Maintainability] One must be able to easily modify a behavior, or even add a new behavioral feature.

[D2: Scalability] The solution should allow scalability since video games usually involve several NPCs.

[D3: Flexibility] The solution must be flexible enough to accept *ad hoc* solutions to specific problems. Indeed, during game development, problems that had not been anticipated often arise.

[D4: Testability] Ideally, the solution should provide testing facilities such as unit testing mechanisms.

[D5: Communication Support] When a large number of people are working on the same project, communication between them becomes critical. A suitable solution should enable efficient communication between those working on a project. It may be a graphical language or automatic generation of a document, for example.

2.1.2 Expressivity Requirements

Several scientific contributions show attempts to define the necessary criteria in order to reproduce human-like behavior. (Livingstone, 2006) categorizes these criteria in three distinct levels: reaction, action and planning. One finds these levels in management and military theories (strategic, tactical and operational decisions). (Tencé, 2011) and (Doherty and O'Riordan, 2006) offer very detailed lists of important criteria to produce believable behavior. However, these lists are designed from basic observations and common sense, and no scientific study (e.g. psychological) measures the link between the proposed criteria and the viewer's perception regarding believability of behaviors. Nevertheless, these criteria proved useful in defining the expressiveness of an AI

solution (Tencé, 2011). The requirements we selected among these lists are involved in the expressiveness of a model. They are summarized in table 2.2.

	Criterion	Summary
Operational	**[E1: Reaction]**	React to players and environmental changes
	[E2: Variability]	Do not reproduce exactly the same behavior again and again. Be somehow unpredictible.
Tactic	**[E3: Understandable]**	Have an understandable behavior
	[E4: Memory]	Be able to memorize information
	[E5: Planning]	Plan actions
	[E6: Evolution]	Do not reproduce mistakes
Strategic	**[E7: Inter Specific Variability]**	Different NPCs should have different behaviors
	[E8: Organization]	Have an understandable role
	[E9: Coordination]	Have coordinated behaviors when in a team

Table 2.2: *List of Requirements for Reproducing Human-like Behavior*

[E1: Reaction] The most basic criteria involved in behavior reproduction is reactivity (Livingstone, 2006). Indeed, NPCs must react to players and to changes in the environment in order to increase the feeling of presence (Tencé, 2011).

[E2: Variability] The way an action is completed has an impact on believability. Very accurate and efficient movements are known to break the illusion of believability (Laird and Duchi, 2000; Livingstone, 2006). Therefore, movements should be as close as possible to those that a real player would do: it is the first thing to see when one faces an avatar. For example, inaccuracy should be added to the movement to create a feeling of humanness (Tencé, 2011).

[E3: Understandable] Unlike realistic characters, believable characters might be forced to exaggerate their behaviors for observers to understand what they are doing (Isla, 2005). Although it can seem strange, it is sometimes important to overemphasize some of the character's characteristics so that people believe them. This technique is quite common in the Arts, especially in cartoons. This means that human-like characters could have quite low believability. However, there are links between realism and believability so one may begin to draw inspiration from realism. We stress the point that believability should be easier to achieve than realism because we do not want the character to exhibit a truly human-like behavior but only to maintain the illusion (Tencé, 2011).

[E4: Memory] It is important that the system can retain information. It may be complex information about the environment or simply remembering its topology. It can also be knowledge about other players, or even the system itself.

[E5: Planning] In order to avoid making counter-productive actions which could break the illusion, the character may need to be able to plan and/or anticipate. By predicting the outcomes of its choices, the character should be able to avoid actions that would lead to easily avoidable mistakes (Livingstone, 2006).

[E6: Evolution] Memory and evolution may be seen as two sides of the same coin, so the character must use evolution mechanisms, for instance by learning (Loyall, 1997, page 20). A character which behaves in the same way repeatedly, even though it is clearly counter-productive, eventually breaks the illusion of being human-controlled. Agents which are able to learn will tend to surprise the players by exhibiting new behavior, adding some unpredictability to the interaction. That is the main reason why believable characters must be able to learn (Tencé, 2011).

[E7: Inter Specific Variability] NPCs can be numerous in video games. Therefore it is crucial that they do not share the same behavior. For example, in a group of players that move together into the environment, if each individual moves along parallel paths at same speed, the group credibility is irreparably compromised.

[E8: Organization] When players work on a collective task, each participant must have an identifiable role.

[E9: Coordination] When players work on a collective task, all participants need to act in a coordinated manner.

2.1.3 Learning Properties

Since we want to learn behavior, several criteria are dedicated to this particular aspect.

Learning properties	Summary
[L1: Online]	Is the algorithm able to proceed incrementally?
[L2: Discrete/Continuous]	Does the algorithm handle discrete or continuous variables?
[L3: Time Series]	Does the algorithm handle time series?
[L4: Real Time]	Is the algorithm fast enough to be processed in real time?

Table 2.3: *List of Requirements for Learning*

[L1: Online] The algorithm should have incremental learning capabilities, that is to say, the data can be submitted downstream to the learning algorithm. If not, is it possible to provide packet data, allowing semi-online learning?

[L2: Discrete/Continuous] What kind of data can the learning algorithm manage: continuous, discrete, both?

[L3: Time Series] Data collected when a human plays a video game are time series. It is therefore likely that including this aspect may allow an increase in the quality of learning. Therefore, does the algorithm treat the data as time series?

[L4: Real Time] Does the algorithm learn quickly? This is a crucial question, since a huge amount of data may be collected during a single player's run. Therefore, it would be useful for an algorithm to be able to be processed in real time on large data sets.

2.1.4 Conclusion

In this section, we have identified the different requirements of a generic software solution for an AI design that would meet the needs of a modern video game. These requirements are divided into three groups: development requirements, expressiveness requirements and learning requirements. In the following sections, these requirements will give us the criteria needed in order to test the models' suitability for our problem.

2.2 Data Mining

In order to reproduce human behavior, we believe that data from real players must be at the center of the method. These data contain a wealth of information that will guide the design of the behavior to be reproduced. However, recovering knowledge within this data is not easy. The discipline is named knowledge discovery in databases (KDD). In this section, we present KDD and show how its techniques can help achieve an AI solution for video games.

2.2.1 Definition

Since the advent of the computer age, the collection of data has become an essential practice in our society. This task already existed in ancient times: the first population census was used to know the number of soldiers available or to determine income from taxation. With the explosion of our data processing capabilities, this collection really took off. IBM estimates that every day, 2.5 billion gigabytes (2.5 exabytes) are collected [1]. But this collection is, of course, not only limited to censuses. Science and industry make great use of this. The use of these huge amounts of data is therefore one of the major challenges of research in the field of computer science. Exploring these amounts of data and extracting knowledge from them is an important challenge. The complexity of the subject requires a multidisciplinary approach that may stretch over many research fields such as AI, statistics or database systems.

In the literature, the term *knowledge extraction* refers to a complex process consisting of several stages. These steps include collecting and preparing the data, the analysis itself and finally interpreting and validating results. Authors such as Fayyad et al. (1996) reduce data mining to the analysis while other authors refer to any of these steps.

The concept of data mining is extremely broad. According to Hand (2007), data mining is to analyze observations (often extensive), to identify unsuspected relationships and to summarize data in a new way that is understandable and useful to their interpretation. These relationships or abstracts often take the form of models or patterns constructed and identified from this data.

2.2.1.1 Data Type

Data mining may be performed on inputs of various size, structure, or form, such as a data table, relational database, graph or a time series. Each data item is generally composed of a set of attributes. The number of attributes defines the size of the data space. Moreover, attributes can be quantitative or qualitative. Quantitative attributes are the attributes associated with a value on which it is possible to perform operations such as calculating a mean or comparison. These attributes can be discrete (population, number of children,

[1] http://www.ibm.com/big-data/us/en/

...) or continuous (height, weight, ...). Qualitative attributes generally correspond to a category (such as car brand, animal species). However, it should be noted that the separation between the quantitative and qualitative is not necessarily trivial. Indeed, qualitative data may be digital (postal codes are numerical but it is not appropriate to calculate an average). In addition, some data can be viewed as qualitative or quantitative. A boolean value for example is essentially qualitative data but by associating false to zero and true to one, it is possible to perform an average with a usable direction. Similarly, analysts can choose to convert data from qualitative to quantitative, according to the type of information they wish to release. One can for example choose to consider the price or the average power of vehicles of a brand instead of the brand itself.

When working with quantitative data, we must first choose how they are represented. This choice of the representation space can be crucial later. For example, choosing to represent a size in centimeters or meters will change the order of magnitude of some results, including calculations of distances. Moreover, it is often appropriate to select or construct a distance different than the classical Euclidean distance. For example, in computer systems, a color is usually represented by its coordinates in the sRGB space defined by Hewlett-Packard and Microsoft in 1996 (Stokes et al., 1996). But it seems more interesting to use a color space like CIE Lab (McLaren, 2008) for representing the colors so the Euclidean distance in this space is more suitable for color perception of most humans.

In data mining problems, it is common to work with incomplete data, for which the attributes may not be complete for some individuals. This is usually the case when working on data aggregation from sources whose collections were carried out under different conditions.

Time series are another example of the type of data that can be processed. In meteorology for example, temperature or humidity data is located geographically, but also temporally. Useful information does not lie only in the value of each parameter but also (and especially) in their evolution.

2.2.1.2 Objectives and Means

Data mining can be considered from two different angles. It can be categorized based on the objectives to be attained or means that you want to use.

Regarding the goals, some authors, like (Fayyad et al., 1996), distinguish two main types: descriptive and predictive. More precisely, we can describe these objectives as follows:

- Prediction: this is a machine learning technique. The goal is to predict the value of a variable (called explained) depending on other variables (called explanatory). The learning process is performed on example data.

There are generally classifications and regressions. Classification is a special case of regression. Indeed classification is for predicting a discrete and finite variable while the term regression is for quantitative variable predictions.

- Data clustering: the key to good clustering is to maximize intracluster similarities and minimize intercluster similarities (Chen et al., 1996). This method involves dividing data into homogeneous groups. There are many applications of this approach. It may be identifying consumer subpopulations forming a homogeneous group. Search engines are another good example. When a user searches for the word "python", a clustering algorithm executed on the results of the request will be able to identify two homogeneous groups of pages: a page group containing computer science content and the other containing animal content. The search engine can then ask the user to refine his request. This is an unsupervised method.

- Automatic Summary: this is a descriptive approach to find a compact representation of the data.

- Modeling dependencies: this consists of identifying dependencies between variables. This allows for example to see that a pathology is significantly more likely to occur in a particular population group. The discovery of association rules (Agrawal et al., 1993), widely used by supermarket chains and online shopping sites, also falls into this category. It is used to identify products that are usually sold together. This information enables the supermarket to shelve and organize their stocks and online stores to suggest relevant products to their clients.

- Outlier detection: in real data, it is common to find data that differ significantly from the others. There can be many reasons for this. It may be an error in the data collection process, so detection of such data can be used to remove them, to fix data collection procedure or replace faulty sensors. It may also be a deviant behavior and outliers then become relevant information. These data may represent fraud (credit card, social security, ...) or an intrusion attempt on a network for example.

Analysts use many tools to perform data mining. Such tools may use different approaches. The interaction between the analyst and the tool defines three types of exploration method:

- Exploratory analysis: the analyst does not necessarily know what to look for in the data. The tool then provides various display facilities. Here, intelligence is not artificial; it is located on the user side. The tool is only in charge of presenting the data to the analyst. However, data visualization is far from easy. Indeed, when data has a relatively small number

of dimensions, it is easy to show a cloud of points representing data. In contrast, when the dimensions of the data increase, this kind of representation becomes impractical. In section 2.2.2, we study different data representation possibilities and techniques for dimensional reduction.

- Supervised analysis: the analyst guides the process by providing the tool with examples of patterns or rules to identify.
- Unsupervised analysis: the tool is solely responsible for identifying patterns in data.

2.2.2 Exploratory Analysis

Exploratory analysis, in the context of replicating human behavior in a video game, can be a very interesting way to understand how players act in a game. Such an analysis can identify knowledge that was not expected beforehand. It may help to identify such invariants in the behavior of the players for the behavior to be [E3: Understandable]. Likewise, we can also identify interesting differences between the players ([E7: Inter Specific Variability]) or homogeneous groups of players.

2.2.2.1 Data Visualization

The main challenge of exploratory analysis is how to present the data. There are many ways to present data and the simplest is probably the data array. Spreadsheet tools are very common for data analysis. Many public databases like UCI[2] provide CSV[3] files which are simple text files presenting data in tabular format. They can be loaded by any Spreadsheet tool. This software can also be used to perform various operations on the data such as sorting, filtering, calculations, or graphical production. A spreadsheet is already a general data mining tool. But more elaborate tools, developed specifically for this task exist, such as R (R Developement Core Team, 2008), ELKI (Achtert et al., 2008), WEKA (Witten and Frank, 2005), Orange Data Mining (Demšar et al., 2013) or GGobi (Swayne et al., 2004). All offer different ways to represent data.

Univariate analysis can be performed using charts presenting data, attribute by attribute (histograms, pie charts, box plots, etc). The diversity of these representations often allows the analyst to find a compromise between the amount and the compactness of information displayed. While histograms provide much more information, box plots provide a more stripped representation. Although those representations are simple and purely descriptive, these univariate analysis already provide highly relevant information about the data.

[2] http://archive.ics.uci.edu/ml/datasets.html
[3] Comma-Separated Values

Such an analysis can sometimes identify a subset of attributes sufficient for performing operations like classification or clustering.

However, a multivariate analysis is generally required. Even if the most natural way of presenting multivariate data is via scatter plots (the coordinates of a point representing the attribute of the corresponding data), we are generally limited to two dimensions, or to three dimensions by using the perspective representation. Of course, it is possible to push this limit, more or less effectively, by varying the color, size or shape of the points. Google Public Data Explorer[4], for example, provides a display illustrating these methods.

We can imagine many other ways of displaying other dimensions, like adding a temporal dimension (point animations, blinking, oscillations, etc.) or adding other sensory information such as sounds. But although these techniques can provide results in some cases, the cognitive load of the analyst remains high and it is very difficult to analyze very large datasets using these techniques. Even 3D representations are hard to interpret without adding interactivity.

We can also choose comprehensive methods such as radar plots or parallel coordinate plots. This, then, allows a clear view of all data attributes, the price of the introduction of an order, a priori arbitrary, in the attributes.

2.2.2.2 Data Dimensional Reduction

Some data processing algorithms can be used to visualize data with large dimensions by performing dimensional reduction. Principal Component Analysis (PCA) transforms a set of potentially correlated variables into linearly uncorrelated variables. Output variables are sorted in descending order of variance. By keeping only the first variables (the principal components), dimensional reduction is performed. As components farther down the ordered list explain the least variance in our data, the distance between each data point is more or less satisfied when they are removed. PCA is a linear method. A nonlinear variant called Kernel PCA (Schölkopf et al., 1998) was proposed. It uses the so called "kernel trick" also used in Support Vector Machine (SVM) presented section 2.2.3.2.

Multidimensional scaling is a technique which, from the matrix of distances between each datum, builds a representational space for our data that maintains these distances. Figure 2.1 shows an example of dimensional positioning achieved by classical MDS proposed by (Torgerson, 1952). This method gives results similar to PCA but like any MDS technique, uses a distance matrix as input. This is a linear method. Many non-linear algorithms have been

[4] http://www.google.com/publicdata/directory

	Bayonne	Bordeaux	Brest	Lille	Lyon	Marseille	Montpellier	Nice	Paris	Perpignan	Strasbourg
Bayonne	0	166	593	866	561	554	432	705	666	367	910
Bordeaux	166	0	496	700	437	506	381	638	500	368	760
Brest	593	496	0	600	765	951	837	1045	506	856	903
Lille	866	700	600	0	558	835	784	834	204	885	409
Lyon	561	437	765	558	0	278	252	299	392	376	383
Marseille	554	506	951	835	278	0	126	160	662	213	616
Montpellier	432	381	837	784	252	126	0	273	596	130	629
Nice	705	638	1045	834	299	160	273	0	686	373	543
Paris	666	500	506	204	392	662	596	686	0	688	398
Perpignan	367	368	856	885	376	213	130	373	688	0	757
Strasbourg	910	760	903	409	383	616	629	543	398	757	0

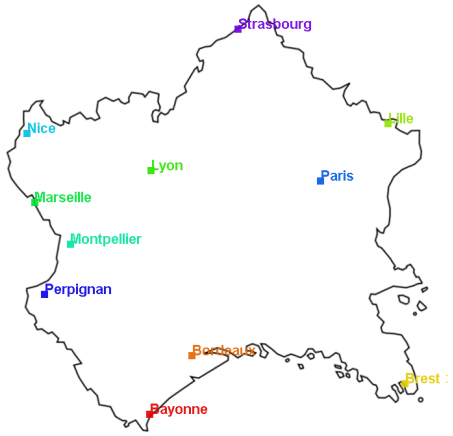


Figure 2.1: Classical MDS Example
Top: Distance matrix provided as the input of the algorithm
Bottom: the reconstructed coordinates of some French cities superimposed with a scaled, rotated and mirrored map of France

proposed, such as Isomap (Tenenbaum et al., 2000), Locally linear embedding (Roweis and Saul, 2000), Diffusion map (Coifman and Lafon, 2006), etc. These methods are designed to perform manifold unfolding; i.e., remove the internal model of the data. Figure 2.2 illustrates such a transformation. In this figure, the Isomap algorithm was applied to a set of images representing the letter A with different rotations and scales. Each image has a dimensionality of 225 (15x15 pixels), but is a constraint on a manifold of much lower dimension. Isomap, here, successfully retrieves the only 2 parameters varying in the data (scale and rotation).

Isomap

Isomap (Tenenbaum et al., 2000) is a dimensional reduction algorithm that, unlike techniques such as principal component analysis or the classic dimensional positioning, can be used to discover non-linear structures in the data.

Isomap (continued)

This algorithm performs three steps:

1. Construct the neighborhood graph: This step consists of building a graph by connecting data i and j of the dataset if the distance $d(i,j)$ is below a value ϵ (ϵ-Isomap), or if datum i is part of k nearest neighbors (see section 2.2.3.2) of the j datum (k-Isomap). The length of the edge is then $d(i,j)$.

2. Compute shortest paths: This step consists of building the distance matrix D between each datum. The required distance is the shortest distance in the graph built in step 1. This step can be performed with various algorithms such as Dijkstra, A* or Floyd-Warshall.

3. Perform dimensional reduction: This step consists of performing classical MDS using matrix D computed in step 2.

Because multi dimensional scaling techniques rely on a distance matrix, they are not able to process new data [L1: Online]. However, some extensions of some such algorithms have been proposed to provide that ability (Shi et al., 2005).

Sammon Mapping

Sammon mapping (Sammon, 1969) is another nonlinear dimensional reduction algorithm. The error function $ERROR_{Sammon}$, given below, is minimized using an iterative algorithm such as gradient descent, as is the case in the article providing the method. This error function marginalizes distance errors in the output space when the distances in the original space are important.

The error function is:

$$Error_{Sammon} = \frac{1}{\sum_{i<j} d^*_{ij}} \sum_{i<j}^{N} \frac{(d^*_{ij} - d_{ij})^2}{d^*_{ij}}$$

where d_{ij} represents the distance between i and j datum in the output space and d^*_{ij} the distance between i and j datum in the original space.

2.2.3 Supervised Analysis

Supervised learning is a machine learning technique to infer a function from labeled data. Formally, it approximates a mathematical function from a space of independent variables to a dependent variable:

$$f : E_1 \times \cdots \times E_n \to Y$$

We distinguish classification, where the dependent variable is discrete (even binary) and regression, whose dependent variable is continuous.

In the case of learning a human behavior, it is clear that this task can be very useful, for example, to reproduce a purely reactive behavior made naturally by the player (target an enemy), but it is not sufficient. A human does not consistently exhibit the same mathematical functions in everything he does.

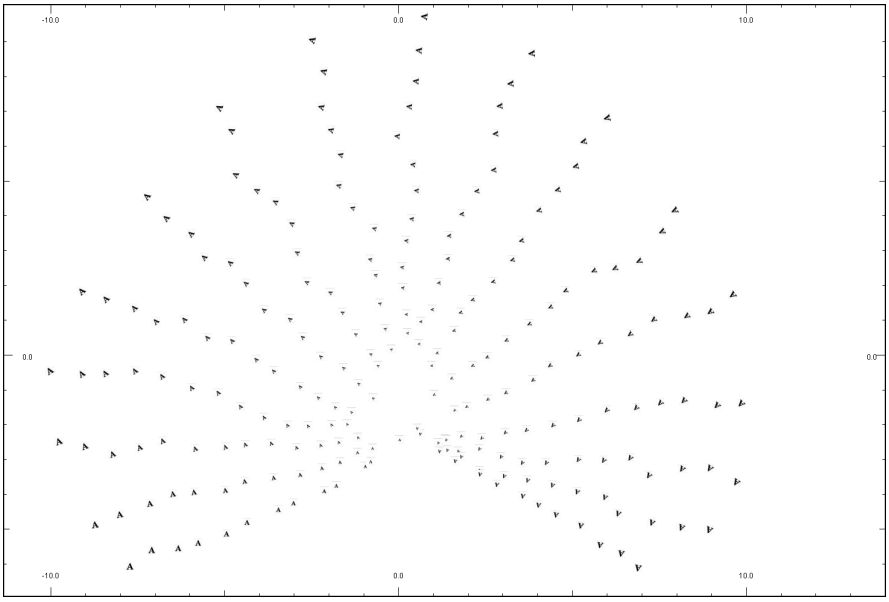


Figure 2.2: Isomap example: Images are rearranged in space as a result of an Isomap algorithm

These techniques will be particularly useful therefore only to reproduce the basic behavior of the player. These basic behaviors match the requirements [E1: Reaction] and [E3: Understandable]. Moreover, if learning is conducted online, the behavior will be able to evolve ([E6: Evolution]).

2.2.3.1 Connectionist Methods

Multilayer Perceptron

The multilayer perceptron is the most well-known type of neural network. In this model, the network is composed of successive layers of which all the neurons are connected to all neurons of the next layer. The first layer (input layer) has as many neurons as there are inputs and the last (output layer) consists of as many neurons as there are outputs. The neuron output is calculated as follows:

$$\phi(w_0 + \sum_{i=1}^{n} w_i x_j)$$

where ϕ is an activation function (such as the sigmoid function or a hyperbolic tangent function, for example), x_i the input data, w_0 the bias of the neuron and w_i the weights of the neuron associated with each entry. The outputs can therefore be calculated, layer through layer, thanks to the input values.

Learning weights of neurons is then generally performed by the backward propagation of errors algorithm.

Connectionist approaches use the biological paradigm of the neuron. These neurons are connected by synapses. In these bio-mimetic systems, these neu-

rons are usually modeled by a linear combination of the inputs to which an activation function is applied. There are however other types of neuron models, such as radial based neurons, wherein the input vector is subtracted from the neuron weight vector for calculating a norm. This norm is then applied as an input to an activation function.

Alone, these neurons are not able to learn nonlinear functions or classifiers. Therefore they are usually used in a network. Multi layer perceptrons are widely used in optical character recognition, for example, where they have obtained excellent results.

Other types of networks exist, including recurrent networks. These networks have a [E4: Memory] effect. The Hopfield networks (Hopfield, 1982) for example, act as an addressable memory.

These connectionist techniques are able to process online ([L1: Online]) but typically require a large number of training samples to produce satisfactory results (there is no [L4: Real Time]). Although not designed for this, they can also be effective on time series ([L3: Time Series])(Koskela et al., 1996).

2.2.3.2 Statistical and Probabilistic Methods

Regression is a term that originates from statistics. It is therefore not surprising that many prediction methods stem from this field. There are two large families of regression methods: Parametric models and non-parametric models.

Linear Regression

Linear regression assumes that the variable is a linear combination of explanatory variables:

$$y = \alpha_0 + \alpha_1 x_1 + \ldots + \alpha_n x_n + \epsilon$$

where y is the explained variable, x_i are explanatory variables, ϵ is the prediction error and α_i the desired parameters. The so-called least squares method was developed in the early nineteenth century and is used to find the parameters α_i minimizing, as its name suggests, the squares of the error of each point of the sample to the model. An extension of this method can be used to find the parameters of a non-linear model.

With parametric models, we assume that data are derived from a model with parameters and we try to identify those parameters to best fit the data. For example, linear classification/regression assumes that the data is separable by/arranged on a hyper plane and we try to find the parameters of this hyper plane. In non-parametric models, no assumptions are made about the data structure. The prediction is performed directly from the data. The most commonly used non-parametric method is the K-Nearest Neighbors (KNN) algorithm. This method is very simple. To make a prediction, we find in our data the k nearest neighbors of the input data and then, we combine the outputs of these k neighbors to make the prediction. There are various ways to combine the outputs of this method. Taking the example of classification,

once we found the nearest neighbors of the input datum, each of them can vote for its output class. Then, the most represented class is chosen as output. It is also possible to weight the voting system so that the nearest data has more influence over the prediction. Parametric regression can also be achieved but only on the k nearest neighbors, thus greatly facilitating computation, rather than doing it over the whole dataset. However, it is necessary to define the notion of proximity, that is to say, to choose a distance. The choice of distance can indeed greatly influence the outcome. Appendix A proposes a study on the influence of distance on various algorithms.

Support Vector Machine (SVM) provides an interesting extension to linear classification. It makes it possible to perform non-linearly separable data classification.

SVM

SVM is an extension of linear regression. This method assumes that the unique optimal hyper-plane is defined as the hyper-plane that maximizes the margin between samples. Formally, we define the following function:

$$h(x) = w^T x + w_0, x \in \mathbb{R}^N$$

if $h(x) > 0$, x is from class 1, otherwise it is the class -1. We thus try to find the equation of a hyper-plane writing:

$$h(x) = w^T x + w_0 = 0$$

The distance between sample x_k and the separator hyper plane is:

$$||x - x_k|| = \frac{|w^T x_k + w_0|}{||w||}$$

The maximum margin separator hyperplane is given by:

$$\arg\max_{w,w_0} \left\{ \frac{1}{||w||} \min_k \left[|w^T x_k + w_0| \right] \right\}$$

Without going into details, this problem can be expressed as follows:

$$\arg\max_{\alpha} \sum_{k=1}^{p} \alpha_k - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j l_i l_j x_i^T x_j \tag{2.1}$$

Under the constraints:

$$\alpha_i \geq 0 \text{ , et } \sum_{k=1}^{p} \alpha_k l_k = 0$$

where l_k represents the class (-1 or 1) of the given k.

In order to process cases that are not linearly separable, the kernel trick is used. The kernel trick uses Mercer's theorem (Mercer, 1909). If a function $K(\vec{x}, \vec{y})$ is continuous, symmetric and positive semi definite, then, there is a space wherein: $K(\vec{x}, \vec{y})$ is the inner product $f(\vec{x}) \cdot f(\vec{y})$, where the function f is a function to project the data in that space. One can show that for functions fulfilling these conditions, the space concerned is of much higher dimension compared to the starting

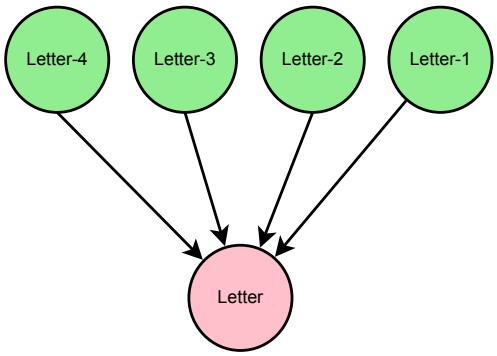


Figure 2.3: Bayesian network used to generate text in different languages

SVM (continued)

space and can even be of infinite dimension. Noticing in the equation 2.1 that $x_i^T x_j$ represents the dot product of two vectors, we can replace this factor in the equation by $K(x_i, x_j)$. The linear separation will be carried out in a much larger space than the size of the starting area. And since there is (almost) always a separator hyper-plane between $n+1$ data in a space of dimension n, it is possible, by choosing well the K function to achieve the separation of our data.

All these statistical parametric regression methods require access to a relatively large amount of samples. The larger the sample, the better the classification/regression. But learning cannot be done [L1: Online]. Moreover, most of these methods are relatively expensive in terms of computation (no [L4: Real Time]).

Bayesian inference is a probabilistic method to assess the probability of an explained variable, given the explanatory variables. It is based on a random variable dependency network. In the most simple networks, it is considered that all predictors are independent and the explained variable is dependent on all the explanatory variables. As an example, we built a network modeling appearances of letters in a text. We consider the random variable corresponding to the n^{th} letter in the text which depends on the previous four letters. Figure 2.3 presents this network. We then evaluate the probabilities of the network from texts in French, English and Romanian. We perform a random draw of an entire text from the network in accordance with the calculated probability distributions. Figure 2.4 shows the results. It is noted that, obviously, the texts do not mean anything, but a reader who does not know one of the languages may have the impression that the text is meaningful. These techniques could be applied appropriately for applications where the semantics are less significant than in a text.

In this example, we assume that the explanatory variables are all independent and known and that the explained variable is directly dependent on all the explanatory variables. In many cases, these assumptions are false. It is then necessary to build a network that models the given problem. The con-

English:

> *so shall daub her flowerets with her flower own children's blood nor mothers' womb to tell beseeming war, like this dear expedienced in for our purposed feet now, under whose soldier now. then let me heaven, all now, under whose no more shock and furious close blessed eyes, what yester.*

French:

> *la splendent sons ta cri rempli de roulis coucher, ni fait les travers que cervent, aimer à la race hume à loisin cieux, destige! le beau comme une peut des marcher. sont-ils qui nous vivant tour, les sens. un amour! nage des ténèbres fonde! n'es pants se commes dont les esprit ont tendre.*

Romanian:

> *cu drept cu fața unui o lacrimă -neiubiră. ei în clasele fără îndoială a aceste idealele frumos îmbătăcită, mai rău zidite și bustul nu mai rău... ce timpul lui de mindependență era orfan, o fiică câte-o femere înălțau ochii cei albenă la un visătoare o palid, multe la aromanțios? adesea sfârșitul.*

Figure 2.4: Example of texts produced by a Bayesian network

struction of this network is usually the responsibility of an expert, but some algorithms are able to learn the network structure (Friedman, 1998).

To handle time series, we generally use dynamic Bayesian networks such as the Hidden Markov Model (HMM) or one of its extensions like Input-Output Hidden Markov Model (IOHMM) (Bengio and Frasconi, 1995). This dynamic Bayesian network is widely used in many fields to analyze time series (speech recognition, financial analysis). The network has a hidden state that depends on its previous state (so, there are probabilities to transit from one state to another). Observed variables then depend on this current state (for a given state, there are probabilities to emit an observable output). Figure 2.5 shows a fairly simple example. Imagine an ice cream shop operating in the subway; the seller does not know the weather outside because he is underground all day. He only knows the number of ice creams he sells per day. This problem can be modeled with a HMM.

There are three major tasks one can perform with HMM. First, knowing the HMM parameters (transition and emission probabilities) and the observations, one can compute the most likely sequence of hidden states with the Viterbi (1967) algorithm. Within this example, the seller only knows that he sells three ice creams on the first day, three on the second and two on the third (the observation sequence is then (3,3,2)). The Viberbi algorithm computes the most likely sequence of weather (it may be (Sun,Sun,Sun)). The second task is, given the parameters of the HMM and the observations, to compute the probability of each hidden state and of the overall observation sequence. This

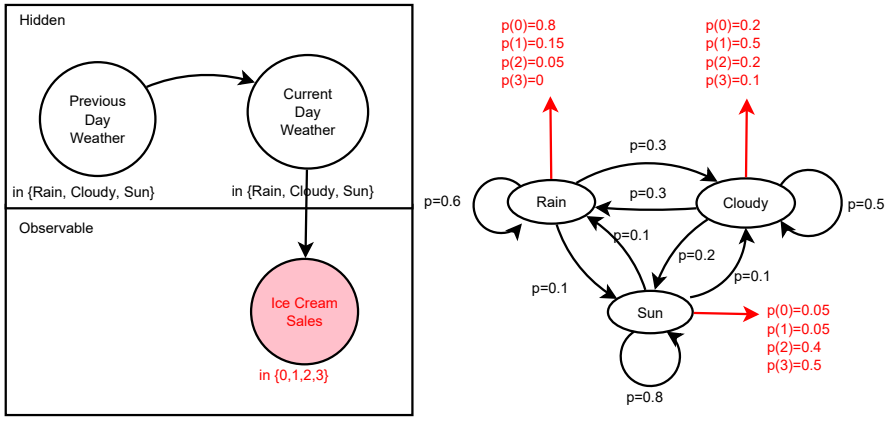


Figure 2.5: HMM example

can be computed with the "Forward Backward" algorithm (described in Russel and Norvig (2009, p.566)). The third task is, given the observation sequence, to learn the HMM parameters (transition and emission probabilities). This is then performed by the Expectation-Maximization algorithm (EM) and, in particular, with the Baum-Welch algorithm (BW) (Baum et al., 1970). Without going into detail, this iterative algorithm maximizes (locally) the probability of observing the sequence of observations. When there are several sequences one wishes classified, it is possible to learn the parameters of a HMM for each class. The classification is then performed by calculating the observation probability of a sequence for each learned HMM and retaining the one with the highest probability.

These probabilistic methods have many advantages: they provide unpredictability [E2: Variability], many algorithms have [L1: Online] implementations, [L4: Real Time], and models exist to efficiently process time series ([L3: Time Series]).

2.2.3.3 Boosting Methods

Boosting methods are used to construct an arbitrarily accurate classifier from weak classifiers, which have slightly greater accuracy than chance. To illustrate the principle, (Freund and Schapire, 1997) offers the example of a horse racing gambler. The latter, frustrated by losing continuously, decides to bet based on the bets of his friends. These friends are here considered as weak classifiers, that is to say that these predictions are a little better than chance. The purpose of the method is then to combine these so-called weak predictions to build a more accurate prediction. AdaBoost (Freund and Schapire, 1997) is probably the most commonly-used algorithm of this category. In this article the authors present an algorithm to solve the problem of binary classification as

well as extensions for multi-class classification and regression. It is successfully used in the Viola Jones method Viola and Jones (2004) for face detection in images.

AdaBoost

AdaBoost (standing for adaptive boosting) (Freund and Schapire, 1997) is a Boosting algorithm based on the selection of a weak classifier to build a strong classifier. In order to achieve this, the learning algorithm uses a weight list $\vec{w}$ quantifying the difficulty of each example on each iteration. Each weight is initialized to $\frac{1}{N}$, where N is the number of examples. At each iteration, we look for the best weak classifier h_t and calculate the error ϵ_t (the sum of the weights of misclassified examples). A coefficient β_t is calculated with this error providing the participation of h_t in the final classifier. The weights of each sample are then updated (they are increased where the classification of the example was incorrect and decreased otherwise). Most of the proposed enhancements of the algorithm are different methods to calculate the relation β_t or to update the weights. The algorithm presented in Figure 2.6 presents the method proposed by (Freund and Schapire, 1997).

inputs
$< (x_1, y_1), \ldots, (x_N, y_N) >$: labeled examples
$\mathcal{H}$: a set of weak learner
T: number of iteration
Init $w_i \leftarrow \frac{1}{N}$
for t = 1 to T **do**
Normalize $\vec{w}$ (Make $\sum_{i=1}^{N} w_i^t = 1$)
Find $h_t, \epsilon_t \leftarrow argmin_{h \in \mathcal{H}, \epsilon}(\epsilon = \sum_{i=1}^{N} w_i |h(x_i) - y_i|)$
Calculate $\beta_t \leftarrow \epsilon_t / (1 - \epsilon_t)$
Update weights: $w_i^{t+1} \leftarrow w_i^t \beta_t^{1-|ht(x_i)-y_i|}$
end for
outputs
the strong classifier:
$h(x) = \begin{cases} 1 \text{ if } \sum_{t=1}^{T} log(1/\beta_t) h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} log(1/\beta_t) \\ 0 \text{ otherwise} \end{cases}$

Figure 2.6: AdaBoost algorithm

2.2.4 Unsupervised Analysis

Unsupervised learning, while learning human behavior, achieves three goals:

- Perform variable discretization. It may be useful, for example, to use supervised algorithms which only support discrete data as inputs, with continuous data ([L2: Discrete/Continuous]),
- Automatically identify homogeneous segments in the data (clustering of [L3: Time Series])
- Perform vector quantization. This can be used to produce knowledge about the environment or opponents for example. In this way, information can be memorized ([E4: Memory]).

2.2.4.1 Centroid Based Methods

K-Means

K-Means produces k centroids (points in the data space) and the point groups associated with each centroid $P_i \in E$ defined by the Voronoï cell of the centroid:

$$Voronoi(P_i) = \{p \in E, d(P_i, p) < d(P_j, p), j \neq i\}$$

The basic algorithm (Lloyd, 1982) consists of:

1. Randomly choosing k centroids in search space ;
2. Assigning each data to its closest centroid (clusters) ;
3. Moving centroid, computing the mean of points within the cluster ;
4. Repeating steps 2 and 3 until convergence.

The k-means algorithm is probably the most commonly-used data clustering algorithm (Jain, 2010). This algorithm, named by (Macqueen, 1966), is a centroid based algorithm. In such algorithms, clusters are defined by the Voronoi cell of centroids.

This very simple algorithm, however, has many shortcomings: it converges to a local minimum, it is very sensitive to the centroid initialization, the desired number of clusters has to be specified, found clusters are convex and linearly separable.

There are solutions to some of these problems. In particular, many articles deal with how to initialize the algorithm, like k-means++ (Arthur and Vassilvitskii, 2007), in which the centroids are selected with a probability that is proportional to the distance to the centroid previously selected. The k-medoids algorithm (Kaufman and Rousseeuw, 1987) imposes the centroids to be part of the input data, which in some cases improves the robustness of the algorithm. The $k$ernel k-means (Zhang and Rudnicky, 2002) (Dhillon et al., 2004) allows no linearly separable clusters. This technique uses the kernel trick also used in SVM.

These centroid based methods allow to perform both discretization and vector quantization and they converge fast enough [L4: Real Time]. However, they do not allow [L1: Online] learning.

2.2.4.2 Connectionist Methods

As we already explained in section 2.2.3.1, many neural network models exist today, but their use is rather suitable for classification or regression. Traditional multilayer perceptrons (MLP) (generally associated with back propagation of the error gradient algorithm) are very good classifiers and are also capable of regression. But they can also be used to perform data clustering. (Charalampidis and Muldrey, 2009) propose, for example, an algorithm which is fairly similar to k-means in its principle, with the difference that the clusters

are no longer represented by the Voronoi cell of a centroid but with the one of a hyper-curve or hypersurface described by the multilayer perceptron.

Most common artificial neural networks to perform unsupervised learning are rather Self-Organizing Map (SOM) (Kohonen, 1982) (Kohonen, 1990). These consist of a fixed network topology (often a grid). The data is submitted to the network. The weights associated to the nearest node of the data are then modified in order to "get closer" to the latter. Neighboring nodes of the nearest node are close together. But the SOM requires the network topology to be predefined. *N*eural Gas (Martinetz et al., 1993) and *G*rowing Cell Structures (Fritzke, 1994) or the Growing Neural Gas (GNG) (Fritzke, 1995b) (Fritzke, 1995a) overcome this problem. Using these approaches, non-convex clusters, and the structure of each cluster (Canales and Chacón, 2007), can be found. The Stable Growing Neural Gas (SGNG) (Tencé et al., 2013) further proposes an adaptation of GNG to better manage [L3: Time Series]. These algorithms are iterative and therefore allow [L1: Online] learning.

2.2.4.3 Hierarchical Methods

The result of clustering can also be seen as a tree structure: a cluster can usually be repartitioned into several clusters. To do this, one has a choice between two approaches: the bottom-up approach (or agglomerative) and top-down approach (or divisive). In the agglomerative approach for a set of N data, we begin with N clusters and group them according to their similarity until a single cluster remains containing all the data. The result of clustering is a dendrogram (tree structure). This method is quite expensive. The key is therefore to carefully choose the similarity criterion to be used for consolidation (this point is discussed in appendix A.3). Some special cases can be handled more efficiently. In the SLINK algorithm (Sibson, 1973), the two clusters with the smallest minimum distance between of two of their points are grouped together. In the CLINK (Defays, 1977) algorithm, the two clusters with the smallest maximum distance between two of their points are combined with each iteration. Both algorithms run in the $\mathcal{O}(n^2)$, where n is the number of data. Other similarity criteria can be used, such as in the CHAMELEON algorithm (Karypis et al., 1999), offering a measurement of the similarity of clusters based on two relative distances, calculated from a k-neighborhood graph.

The top-down approaches consist of starting from a single cluster that contains all the data and dividing it successively. This method, although less common, may have, as Guénoche et al. (1991) argue, several advantages over the agglomerative approach. The most obvious argument is that, in most cases, we try to obtain a small number of clusters. The number of iterations of an algorithm which proceeds by divisions will be generally lower than the number of iterations of a agglomerative algorithm.

2.2.4.4 Probabilistic Methods

Gaussian Mixture Model (GMM)

Gaussian Mixture Models (GMM) is a probabilistic model that can be used to cluster data. In this model, it is assumed that each sample data item has been produced by a law which consists of a linear combination of Gaussians.
For a mixture of k Gaussian (univariate in the later for simplicity), this law is defined as:

$$f(x) = \sum_{i=1}^{k} \lambda_i G(x, \mu_i, \sigma_i)$$

where λ_i are mixture weights, G the Gaussian law, μ_i and σ_i the mean and standard deviation of the Gaussian.

The Gaussian parameters are generally determined by the Expectation-Maximization algorithm (EM). This iterative algorithm attempts to maximize likelihood. It performs the following steps:

- Randomly initialize the parameters λ_i, μ_i et σ_i of each Gaussian
- While the algorithm has not converged:
  - For each Gaussian and for each data, compute γ_{ij} the probability of the i-th datum to had been produced by the Gaussian j
  - μ_i=weighted (by γ_i) mean of the data
  - σ_i=weighted (by γ_i) standard deviation of data
  - $\lambda_i = \sum_j \gamma_{ij}$

Thus, after the execution of the algorithm, it is possible, for each point, to calculate the probability that it is produced by each Gaussian, defining the partitions.

Unsupervised probabilistic methods usually involve supposing that the data came from a statistical model and finding the parameters of this model. These parameters are usually learned by an EM algorithm. The Gaussian Mixture Model (GMM) is a representative example of this family. Other models have been developed in order to handle [L3: Time Series] such as dynamic Bayesian networks. As an example, HMM, already presented in section 2.2.3.2 can also perform clustering (Smyth, 1997).

2.2.4.5 Density Based Methods

DBSCAN (Ester et al., 1996) and OPTICS (Ankerst et al., 1999) are methods designed to overcome many of the recurring problems with other data clustering algorithms. They detect outliers and their clusters can be of any shape. These are density based methods, that is to say, they require parameters to specify a minimum density for the data not being considered as outliers. They do not, however, perform vector quantization, and they cannot be used [L1: Online].

DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm based on density insofar as it use the estimated density of the clusters to complete clustering. The algorithm finds the ϵ-neighborhood of a datum and, if this neighborhood contains enough points, the area is considered to be sufficiently dense to be a cluster. The data in $epsilon$-neighborhood are traversed and added to the current cluster until there are no more points to go. Once a cluster is finished, the algorithm iterates with another, not yet traversed, datum. The algorithm is as follows:

DBCAN(D, ϵ, M)
inputs
  $D = x_1, \ldots, x_N$: data
  ϵ: distance to consider
  M: minimum points to consider
$ClusterId \leftarrow 1$
for i = 1 to N **do**
  if x_i is Unclassified **then**
    if Expand(D, x_i, $ClusterId$, ϵ, M) **then**
      $ClusterId++$
    end if
  end if
end for

Expand(D, p, $ClusterId$, ϵ, M)
inputs
  $D = x_1, \ldots, x_N$: data
  p: point to expand
  $ClusterId$: current cluster Id
  ϵ: distance to consider
  M: minimum points to consider
$seeds \leftarrow \epsilon - neighborhood(D, p)$
if size($seeds$) < M **then**
  Set p to Noise
  return false
else
  Set p to $ClusterId$
  remove p from $seed$
  while $size(seeds \geq 0$ **do**
    $currentP \leftarrow seeds[0]$
    remove $currentP$ from $seeds$
    if $currentP$ already visited **then**
      continue
    end if
    $results \leftarrow \epsilon - neighborhood(D, currentP)$
    if size($results$) > M **then**
      add all $results$ to $seeds$
    end if
  end while
end if

2.2.5 Conclusion

We have seen how various data mining tasks can be useful in creating AI in video games. For each of these tasks, we have also performed a brief review of existing techniques. All these techniques are interesting but none is useful in every situation. A choice has to be made by the analyst regarding which method to use, depending on the given context. The analyst must choose the best on a case-by-case basis. We also find that the various tasks carried out

using data mining techniques do not meet all the requirements identified in this chapter.

2.3 Discussion

In this chapter, we have put into perspective the needs of the video game industry and different Data Mining and Machine Learning techniques, which mainly come from research. We concluded that Data Mining offers a variety of tools to perform tasks that would be very helpful to develop an AI in video games based on using data from human players. However it appears, firstly, that Data Mining is a complex process involving many disciplines, and secondly, that on its own it is not sufficient to produce complex behaviors required for creating AI in video games.

2.3.1 Data Mining

During our study on Data Mining and the different techniques it offers, we found that many of these techniques can be used to perform many tasks. The KMeans algorithm for example, can be used for clustering and vector quantization. The HMM can be used in many ways. There are many techniques available, developed with multiple variations, and parameters need to be set correctly to produce the expected results. Use of these techniques thus requires rather extensive knowledge of these algorithms.

Modeling these techniques would seem an interesting approach to provide a level of abstraction to potential users. In tools like WEKA, GGobi or ELKI, algorithms are often grouped into categories (Pre-Process, Classification, Regression, Clustering, etc). It seems that a little more refined modeling is possible in particular by grouping algorithms by task (filtering, discretization, clustering, vector quantization, etc) and by offering several levels of abstraction. This task modeling however requires in-depth knowledge of Data Mining.

Data visualization is both a sizable and difficult task. Data can be viewed in several ways (scatter plot, histograms, etc). These common visualization methods are available in the aforementioned tools. But these general methods do not take into account the semantics of data. Consider an example: when trying to make changes or prediction on images, viewing by scatter plot is unnecessary and diagrams such as histograms are not sufficient. For these tools, the imported data are only numerical or categorical attributes. As an example, they cannot take into account the fact that three real values represent a vector and display it in this form. Furthermore, the addition of semantics would allow transformations or calculations that would be relevant only for a particular type, such as calculating an angle between two or three dimensional vectors.

2.3.2 Behavior Production

Since these Data Mining techniques do not fully satisfy the requirements expressed in section 2.1, we must find a way to overcome these drawbacks. These shortcomings are mainly related to the development requirements identified in section 2.1.1. Therefore, a solution appears fairly obvious: to use a global architecture which is possibly already used in the video game industry. This architecture should however be able to use data mining techniques in order to construct behaviours by learning from data produced by human players.

Contents of Chapter 3

Chapter 3

ORION: A Generic Model Proposition for Imitation Learning of Behavior

I saw the inner me.

Andrew - Bicentennial Man (1999)

Summary

Most artificial intelligence techniques used in video games may be integrated into different kinds of global architectures. We study the following types of architectures: procedural, rule-based, structured and multi-agent. We conclude from this study that structured architectures are the best suited to fulfilling our requirements.

We then present the ORION model. This model is composed of a structural part and a behavioral part. The structural part enables datasets to be handled generically and even allows semantics to be added to data. The behavioral part is based on behavior trees as they satisfy many of the requirements expressed in chapter 2. Moreover, we propose an extension that constructs artificial intelligence from data retrieved from human players.

Summary

When using behavior trees, data are usually stored in a key-value dictionary used by every behavior. The structural model helps to streamline inputs and outputs used by the model. Orion *offers typed data, therefore allowing the semantics of data to be maintained throughout the design as well as checking the consistency of the data used by each behavior. Thanks to our implementation of the model, we demonstrate the flexibility of our approach by manipulating and visualizing various data sets from a widely used database known as UCI.*

The behavioral model enables behaviors performed by machine learning, both online (during the execution of the game) and offline (during the design phase), to be integrated within its design. These behaviors can produce additional knowledge from data (such as automatic learning of the environment) or automatically partition the set of sequences in order to identify different strategies used by the player.

Introduction

In the previous chapters, we have identified the necessary requirements of our solution and we proposed to use various data mining techniques in order to reproduce human behaviors in video games. Now we have to integrate these techniques into a more comprehensive architecture dedicated to the design of higher-level behaviors. To do so, we have chosen Behaviors Trees as they fulfill most of our requirements. We propose extensions of this model that overcome the shortcomings that a behavior imitation learning solution requires.

Plan

Since we have seen in the previous chapter that data mining techniques lack some requirements we identified, we need to choose a broader architecture that will integrate data mining techniques. In section 3.1, we study general purpose architecture families for AI.

In section 3.2, we present the structural model of Orion. We start by describing how the data are represented. Data representation is an important feature. Since we want to obtain behaviors from these data, particular attention must be paid in order to generically manipulate and visualize them while keeping their semantics.

In section 3.3, we present the behavioral model of Orion. This is an extension of the behavior tree model. We start by accurately describing this model. Next, we describe the extension of this model that handles behaviors construction by learning from a dataset (behaviors that can be learned online or offline).

3.1 General Models for AI Systems

We have demonstrated, in the previous chapter, the value of using data mining to build our solution. Classification and regression algorithms can reproduce reactive behaviors ([E1: Reaction]) and some of them are able to partly produce variability ([E2: Variability]). The extraction of knowledge can be implemented through vector quantization techniques while providing [E4: Memory] to the solution. When these techniques are used [L1: Online], the learned behavior can get better in time ([E6: Evolution]).

However, these techniques are not, strictly speaking, a technical development that meets the requirements defined in section 2.1.1. In addition, some expressivity requirements defined in section 2.1.2 are not met. [E5: Planning], [E7: Inter Specific Variability], [E8: Organization] or [E9: Coordination] are indeed difficult to learn automatically. Because we want to build a solution able to learn this expressivity, we must implement the use of data mining solution in a more general architecture, responding to development criteria such as [D1: Maintainability], [D3: Flexibility] or [D5: Communication Support].

In this section, we study different families of architectures that can be implemented as part of an AI solution in video games. This study will allow us to choose the most suitable architecture to our requirements.

3.1.1 Procedural Models

Most games are designed using imperative and procedural programming languages. If the first few games directly used the machine language for which they was designed. Very quickly, more advanced languages appeared extending this paradigm. Thus, games are now often created with object-oriented languages. This paradigm allows better [D1: Maintainability] than purely procedural languages. Regarding the implementation of AI, game engines usually provide dedicated script language. For example, the Unreal engine offers UnrealScript inspired by Java, while Lua is a language widely used in many games (it is present, for instance, in games using the CryEngine, World of Warcraft or Fable II). These scripting languages allow a great [D3: Flexibility], since it is possible to design almost any feature by this method. However, [D4: Testability] can be penalized without the use of dedicated tools. The source code, meanwhile, can hardly serve as a [D5: Communication Support] between different development teams.

3.1.2 Declarative Models

While procedural models consist in the description of successive actions that will reach a specific result, declarative methods consist only in the declara-

tion of a set of actions and their effects. This method is used in the game *F.E.A.R.* (see chapter 1). Production systems can then make use of these sets of actions/effects. In this category of methods, one can find for example some cognitive architectures such as Soar (Laird and Duchi, 2000) or programming languages such as Prolog. This information can also be used by automatic planners such as STRIPS. These approaches have the advantage of allowing the addition of new features rather easily, bringing [D3: Flexibility] and [D1: Maintainability]. But they are difficult to test ([D4: Testability]). Since they are based on purely declarative knowledge, they do not necessarily require technical knowledge to be understood and might therefore be efficient as a [D5: Communication Support].

3.1.3 Structured Models

The probably most used architectures for IA development in video games are structured architectures. As we saw in chapter 1, finite state machines are widely deployed in today's gaming systems. While they have been developed to enable better [D1: Maintainability] over procedural models, the complexity of AI makes FSM increasingly complicated and less maintainable. For these reasons, another structured architecture is becoming more and more popular: the behavior trees. In this architecture, the goals and sub-goals are represented as nodes in a tree. But the nodes may be used at several places in the tree, so that this model is more, formally, an acyclic graph than a tree. Anyway, this structure allows better [D1: Maintainability] than FSM. In addition, the leaf nodes usually run the *ad hoc* code offering a large [D3: Flexibility] (although this is also the case in FSM).

UML is a language mainly used for software development. It is a graphical language naturally designed to act as [D5: Communication Support] during software development. It can express the structure of the software as well as its behavioral content, interactions between components or even the integration of the software solution within the material necessary to operate it. The behavioral description tools included FSM but also activity diagrams. These allow to describe and define parallelizable behaviors and complex processes with an acceptable level of [D1: Maintainability]. The UML model is usually filled in tools which generate the source code. But most of these tools only generate the code of the structural part of the model. However, some work, such as the Mascaret library (Querrec et al., 2013) provide an implementation to interpret behavioral models.

All these structured architectures have inherently interesting characteristics: They provide a [D5: Communication Support] between the different actors of development.

3.1.4 Multi-agents Models

In this section, the term *multi-agents* is considered in the broadest sense. We present architectures that take into account other characters, either controlled by humans or not. In the previous section, we mentioned the Mascaret model. This model, in addition to an UML interpreter, provides an extension to it. This extension allows to include information in the model related to the virtual environment (such as spatial constraints (Trinh et al., 2011)), interactions (which objects are interactive, which kind of interactions, etc) and the [E8: Organization] of entities that evolve in the environment. An activity diagram is then used to define how entities have to coordinate ([E9: Coordination]).

The BIGRE model (Rivière et al., 2012) uses a Belief Desire Intention (BDI) architecture to generate facial expressions. Expressions of the agent should react not only to its perception of the environment but also to the perception of other people. Thus, the model adds a social interaction appearance. Such models could be extended to other fields of application than emotional management and thus provide a way to obtain [E7: Inter Specific Variability] by implementing different characters into our AI.

3.1.5 Summary

From the different architectures that we mentioned in this review, we choose not to retain, for our solution, procedural approaches that do not offer enough [D1: Maintainability] and [D3: Flexibility], the most important development requirements to our opinion. Moreover, they do not meet a large number of other important criteria. Multi-agent approaches seem interesting in the development of AI of cooperating NPCs. However, these approaches seem very complex and development-related demands seem difficult to obtain with this kind of model. For these reasons, structured approaches and rules-based approaches seem more appropriate. Declarative models are very interesting because they allow [E5: Planning] by essence, and because they are goal oriented (to obtain [E3: Understandable]). The behavior tree is also goal oriented, but does not allow [E5: Planning]. However, it better fits the development criteria including [D4: Testability]. There is therefore no trivial choice on the possible overall architecture of our solution. Table 3.1 summarizes the concordance of each paradigm requirements. We choose a structured solution and, more particularly, behavior trees as they correspond the best to our criteria.

3.2 ORION Structural Model

The datasets we want to use to learn behaviors can be of various types and forms. The ORION model proposes a generic approach to represent these datasets and enables as well some transformations and their visualization.

Requirement	Procedural	Rule based	Structured	Multi-Agent
[D1: Maintainability]	✗	∼	✓	✗
[D2: Scalability]	✓	∼	✓	✓
[D3: Flexibility]	✗	✓	✓	∼
[D4: Testability]	✗	∼	✓	✗
[D5: Communication Support]	✗	∼	✓	∼
[E1: Reaction]	N/A	N/A	N/A	N/A
[E3: Understandable]	∼	✓	✓	∼
[E4: Memory]	N/A	N/A	N/A	N/A
[E6: Evolution]	N/A	N/A	N/A	N/A
[E5: Planning]	∼	✓	∼	∼
[E7: Inter Specific Variability]	∼	∼	∼	✓
[E8: Organization]	∼	✗	✗	✓

Table 3.1: *Concordance Summary of Model Requirements*

Datasets can be time-series or not, contain quantitative and qualitative attributes, be abstract or have strong semantics. The way our model will represent data must cover all these cases and fill some gaps in the existing software. The data semantics is generally not taken into account in different data processing and representation tools such as ELKI (Achtert et al., 2008) WEKA (Hall et al., 2009) or GGobi (Swayne et al., 2004). Even if these tools allow to visualize data through scatter plots, histograms and other diagrams, in order to perform an analysis, it is essential to visualize the data according to their semantics. The problem is well identified by Vondrick et al. (2013). They put it into focus with object detection in an image using feature extraction techniques. Once feature extraction is done, the data becomes abstract. Investigating why a particular data is misclassified, for example, becomes extremely complex. In order to maintain understandability of data throughout the analysis process, the authors propose an image reconstruction from the extracted features (in this article, histograms of oriented gradient). This is indeed a major problem that must be addressed. Visualization and transformations applied to the data should be consistent with their semantics.

3.2.1 Data Representation

In our model, we preserve the semantics of the data via a generic approach: each attribute has a type. An interface `Type` must be implemented for each type of data to be processed. The ORION model provides basic types (reals, integers, enumerations) but also vectors, images, etc. The components that will transform or display the data therefore have access to the semantics associated. A `Type` is considered as a possible data aggregation. A three-dimensional vector is, for example, considered as the aggregation of three values. These values are necessarily convertible to real numbers. Thus, an image is composed

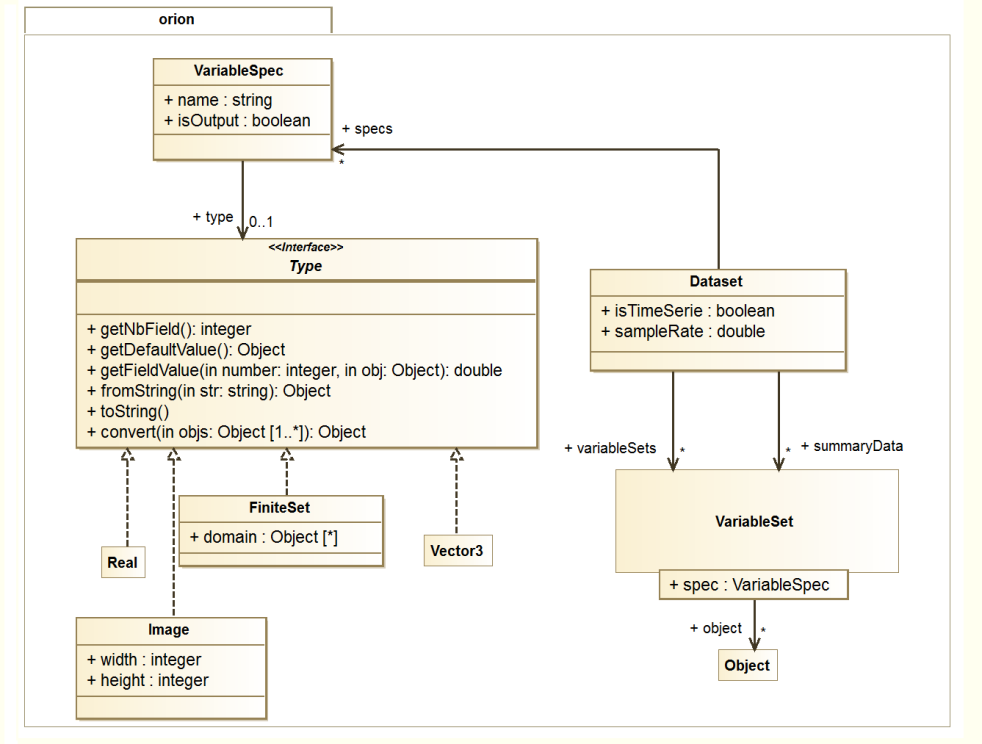


Figure 3.1: ORION data model

of the width × height (× 3 if it is a color image) real numbers representing the intensity of each pixel. An enumeration is composed of an integer (and thus a real) representing its position in a list of possible values of the enumeration. This allows to process data from any type of component even if no processing or display is able to take into account the semantics of the data. The `Type` interface is also in charge of the textual representation of the data. Thus, a user can edit with a simple text field. Finally, this interface is also responsible for data conversion to another type. For example it is possible to convert three real values into a `Vector3`.

A data set is represented in the model ORION by the `Dataset` class. This class stores information on the data set, like for instance its name, or the fact that it is or not a time series, and in that case its sampling frequency, etc. This class is composed of `VariableSpec` representing the specification of each attribute in the data set and `VariableSet` representing a given data set. The `VariableSpec` consists of the name and type of the attribute, and whether the attribute is an input or an output. The `VariableSet` consist of a list containing their values referenced by the `VariableSpec` of the corresponding attribute. Figure 3.1 shows the UML class diagram that implements our data representation. The `Dataset` class is also composed of an attribute

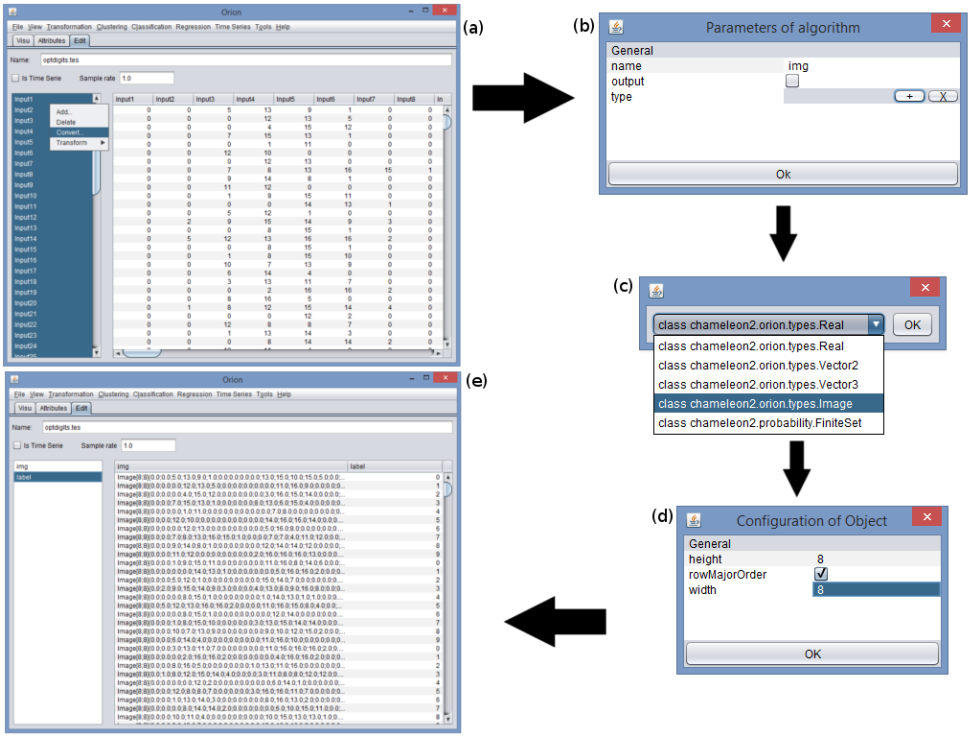


Figure 3.2: Type Conversion process for the Optitrack Database in ORION.

`summaryData` of type `VariableSet`. This attribute is mainly used to store the result of vector quantization for example. Attributes can be converted from one type to another. To illustrate data conversion, we present in figure 3.2 the conversion process of the Optitracks database available on the UCI database[1].

This dataset represents images of handwritten digits. The original image was made up of two colors and had a resolution of 32x32 pixels. Each image was cut into blocks of 4x4 pixels. The dataset consists of 65 attributes. The first 64 attributes represent the number of black pixels in each block of 4 pixels by 4. The 65th represents the class of the data (a digit). After importing the csv file into the tool, the process is made of five steps: (a) selection of the attributes to be converted, (b) specification of the parameters of `VariableSpec` into which the data will be converted (the name of the attribute, input or output), (c) type selection and optionally (d) configure the parameters of this type (here it is the image type, so we need to specify its dimensions). Step (e) shows the result of the conversion of the first 64 attributes in an attribute (which was the type `Real`) to a type `Image` and the 65th (which was of type `Int`) into `FiniteSet` type. ORION includes the following types:

[1] http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

- `Int`: an integer
- `Real`: a real
- `FiniteSet`: a finite set
- `Vector2`: a 2-dimensional vector
- `Vector3`: a 3-dimensional vector
- `RotationVector`: an orientation in a 3-dimensional space
- `Quaternion`: a quaternion
- `Basis3D`: a 3-dimensional localization (position and rotation)
- `Vector`: a n-dimensional vector
- `Image`: an image
- `Graph`: an oriented graph

This list can be extended by implementing the `Type` interface. These implementations are then available in our tool via the reflection mechanism of the language (here, Java).

3.2.2 Data Transformation

In order to perform data transformations, the generic `DataTransform` interface must be implemented. The latter simply has access to the `Dataset` and a list of `VariableSpec` to process and must implement the `DataTransform.transform()` method. Allowed transformations can vary significantly. They generally involve attributes modification, addition of attributes or `summaryData`, etc.

The downside of this simplicity is that the GUI for manipulating this model has no information on the type of transformation performed by the `DataTransform`. In order to overcome this problem, various sub-interfaces are available by ORION:

- `ClusteringTransform`: Classes implementing this interface perform data clustering. That is to say, they add to the `Dataset` an attribute corresponding to a class identified by a partitioning algorithm.
- `SummarizeTransform`: Classes implementing this interface add `summarizeData` to the dataset.
- `DataTransformer`: Classes implementing this interface are able to process new data not belonging to the `Dataset`. It must therefore implement an additional method taking a parameter `VariableSet` (a datum). As an example, the class `NormalizeDataTransform`, which scale real numbers to the [0,1] interval implements this interface.

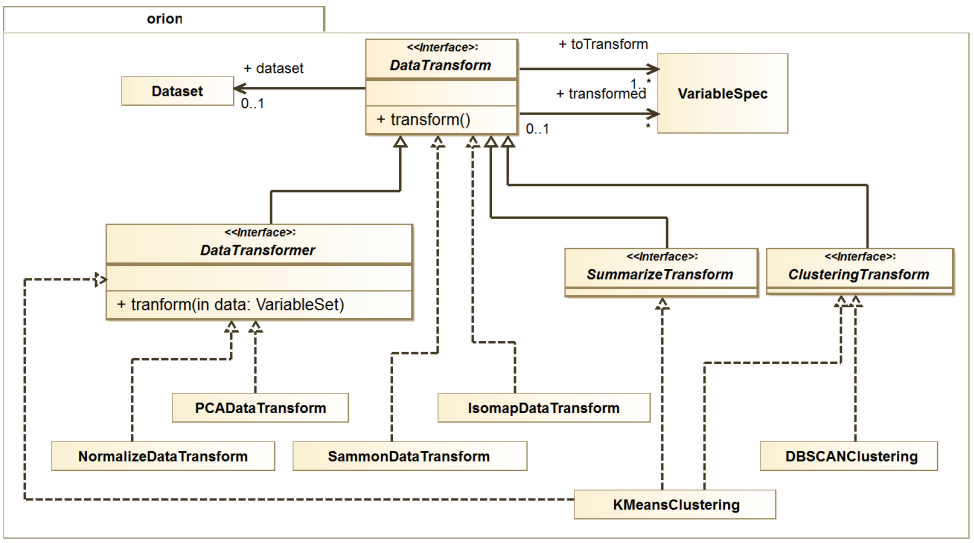


Figure 3.3: ORION data transformation model

- `DiscretizerTransform`: Classes implementing this interface perform discretization of the data.
- `VectorQuantization`: Classes implementing this interface perform vector quantization of the data.

These interfaces are only present to classify data transformation methods into several categories. An algorithm can also implement several of these interfaces. For example, the implementation of the algorithm KMeans in this model implements the interfaces `ClusteringTranform` and `SummarizeTranform`. Figure 3.3 shows the UML class diagram implementing the transformation model of ORION.

Using the example of UCI's Optitracks database, original images used to build the dataset are not available because each image is represented by the sums of black pixels in a block of 4 by 4 pixels. However, it is possible to reconstruct 8x8 grayscale images with this data. To do this, we need to normalize each value between 0 and 1 (1 being white and 0 black). ORION provides a class `BatchDataTransform` to apply an expression to each value of the data set. In addition, it may be useful to produce multidimensional scaling on data. Figure 3.4 illustrates the multidimensional scaling execution process carried out by the Isomap algorithm through our tool. Data normalization was previously carried out via a similar process.

Data transformation algorithms implemented in ORION include:

- `Normalize`: This transformation scales the value of a real attribute in order to normalize it between a minimum and a maximum value, typically

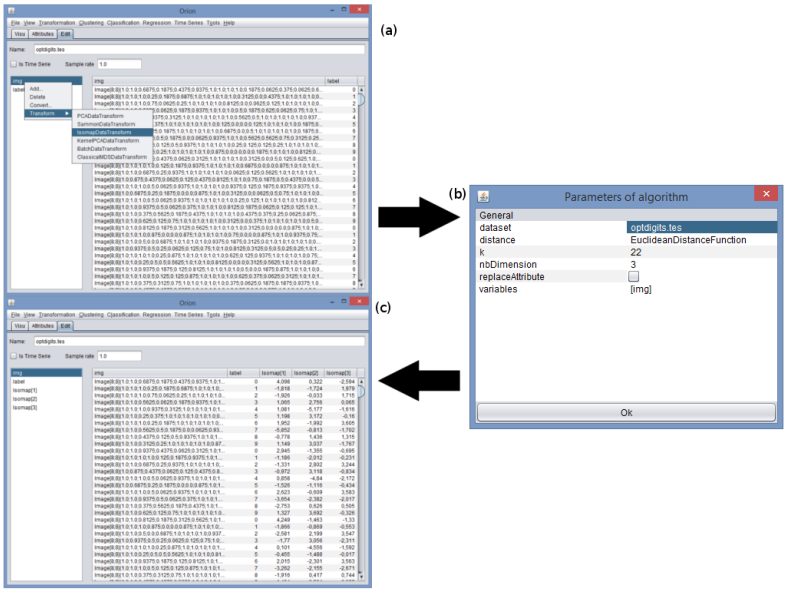


Figure 3.4: Transformation process for the optitrack database (Isomap).

between 0 and 1. This class implements the `DataTransformer` interface.

- `StandardScore`: This transformation standardizes a real variable (subtract its mean and divide by its standard deviation). This class implements the interface `DataTransformer`.

- `Differentiate`: This transformation computes the derivative of an attribute of time series over time. This class implements the `DataTransformer` interface.

- `Batch`: This transformation changes the value of an attribute with a javascript script. Various values are put into the execution context of the script as the `VariableSpec` of the attribute, its current value, its average, maximum and minimum values, etc. This class implements the `DataTransformer` interface.

- `PCA`: This transformation performs a principal component analysis on the data. This class implements the `DataTransformer` interface.

- `KernelPCA`: This transformation performs a principal component analysis using the kernel trick on the data. This class implements the `DataTransformer` interface.

- `MDS`: This transformation allows a classical multidimensional scaling on the data.

- `Isomap`: This transformation allows nonlinear multidimensional scaling using Isomap algorithm.
- `Sammon`: This transformation performs Sammon projection on the data.
- Clustering: These transformations add an attribute to the data set corresponding to the class identified by the clustering algorithm.
  - `KMeans`: This transformation allows data partitioning via the algorithm KMeans. It adds a `summarizeData` to the dataset.
  - `DBSCAN`: This transformation allows data partitioning via the algorithm DBSCAN.
  - `SLINK`: This transformation allows data partitioning via the algorithm SLINK.
  - `CLINK`: This transformation allows data partitioning via the algorithm CLINK.
  - `GMM`: This transformation allows data partitioning through a Gaussian mixture model.
  - `GNG`: This transformation allows data partitioning via the GNG algorithm. It adds a `summarizeData` to the dataset.

This list can be extended by implementing the `DataTransform` interface. These implementations are then available in our tool via the reflection mechanism of the language at use (here, Java).

3.2.3 Data Visualization

In the ORION model, every element in charge of data visualization implements the `DataViewer` interface. This interface is configurable by the enumeration `DisplayRange` to define the data to be represented (the current datum, the current selection or all data). Two display types are present: the 2D and 3D rendering interfaces implements the `GraphicalViewer` interface while the various components displaying data, attributes by attributes, implement the `AttributeViewer` interface.

The `GraphicalViewer` interface is implemented by the classes `Viewer2D` and `Viewer3D`. This interface is not intended to be extended by other classes. A `GraphicalViewer` is composed of a `Renderer`. The `Renderer` interface provides methods (`draw2D` and `draw3D`) to draw a datum on the display panel and a method to provide a dialog panel to configure the `Renderer`. For example, the implementation of `PointRenderer` allows to represent data as points. The position of the point can be defined by an attribute of type `Vector2` or `Vector3` or three `Real` attributes. Similarly, its size, shape or color can be specified as fixed, varying with an attribute value or with its position in the time series. Figure 3.5 shows the control panel of a `PoinRenderer`. A `CompositeRenderer`

simply allows multiple `Renderer` to display the data, each `Renderer` displaying different attributes of the data.

ORION implementations of `Renderer` include:

- `PointRenderer`: This `Renderer` is used to represent data points. It is possible to vary the position, size, shape and color depending on the value of attributes of the data to display.
- `VectorRenderer`: This `Renderer` is used to represent data by lines. It is possible to vary the position, the size, thickness and color depending on the value of attributes of the data to display.
- `ImageRenderer`: This `Renderer` is used to represent data as images. It is possible to vary the position, size and color of a frame surrounding the image according to the attribute value of the data to display.
- `TextRenderer`: This `Renderer` is used to represent data by texts. It is possible to vary the position and color based on the attribute value of the data to display.
- `Base3DRenderer`: This `Renderer` is used to represent a 3D location and orientation.
- `CompositeRenderer`: This `Renderer` is used to combine display of data from multiple `Renderer`.

This list can be extended by implementing the `Renderer` interface. These implementations are then available in our tool via the reflection mechanism of the used language (here, Java). Figure 3.7 shows the display of UCI's Optitracks dataset as well as the display of NAO robot's skeleton taken from a goalkeeper agent in the context of the RoboCup 3D Soccer Simulation Competition.

The interface `AttributeViewer` consists of a list of attributes to display. ORION implementations of `AttributeRenderer` include:

- `RadarPlot`: Displays the attributes as a radar plot
- `ParallelePlot`: Displays the attributes as a parallel plot
- `HistogramPlot`: Displays the histogram of the attributes
- `LineChart`: This component, intended for time series, displays the temporal evolution of each attributes

This list can be extended by implementing the `AttributeViewer` interface. These implementations are then available in our tool via the reflection mechanism of the language at use (here, Java).

Figure 3.6 shows the UML class diagram that implements the visualization model of ORION.

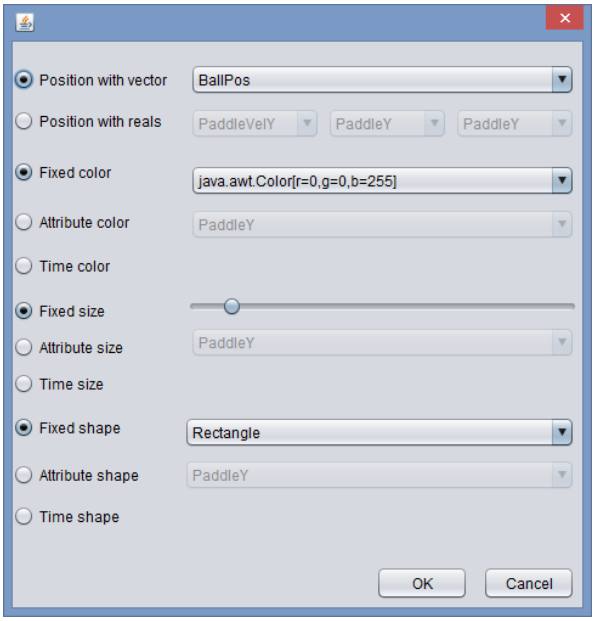


Figure 3.5: Config dialog for a `PositionRenderer`

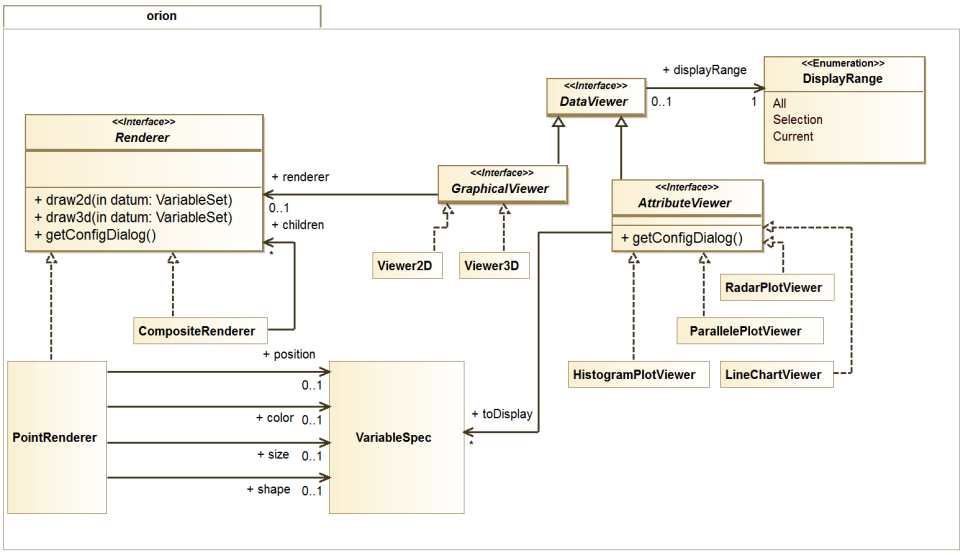


Figure 3.6: ORION Data Visualization Model

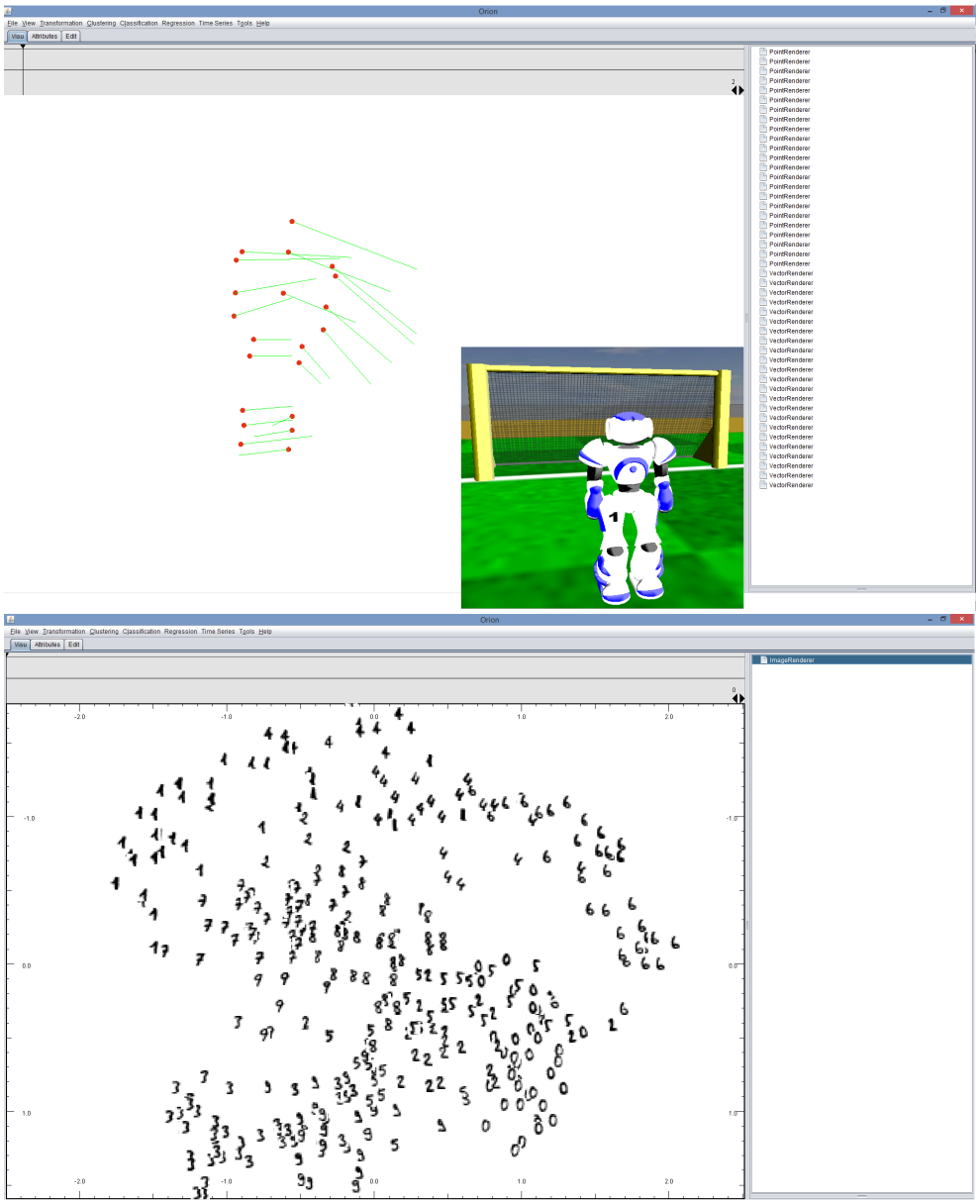


Figure 3.7: Example of datasets visualizations in ORION.
Above: NAO Skeleton dataset (bones positions and speed). The picture at the bottom right, that is not part of ORION GUI, illustrate NAO within the simulation.
Bottom: UCI Optitracks dataset

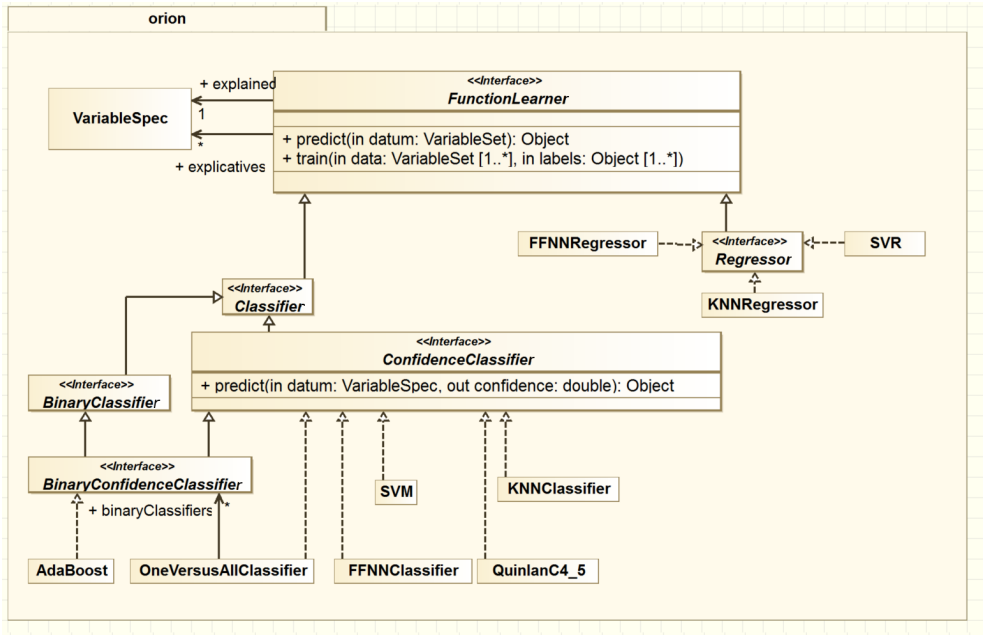


Figure 3.8: ORION data prediction model

3.2.4 Data Prediction

Data prediction is an important task of our data-based learning approach. In order to achieve data prediction and therefore improve learning functions, the model ORION offers a level of abstraction for all methods of classification and regression. Figure 3.8 illustrates this model. The `FunctionLearner` interface represents a regression or classification algorithm. This interface is composed of the explanatory variables and the dependent variables. It has the methods `predict` and `train` that are respectively used to predict the dependent variable based on explanatory variables and to train the algorithm based on sample data (containing the explanatory variables and the corresponding dependent variable). The `Classifier` and `Regressor` interfaces extend `FunctionLearner`.

Several interfaces extend `Classifier` to separate different characteristics of these algorithms. Indeed, many classification algorithms are binary classifiers (although most classification methods being initially binary have seen multiclass extensions proposed like for SVM and AdaBoost). In addition, many algorithms do not perform a peremptory classification but instead return a probability of belonging to each class. Interfaces `BinaryClassifier`, `ConfidenceClassifier` and `BinaryConfidenceClassifier`, therefore express these features. Thereby, it becomes possible to implement the standard methods of extending of the binary classification problem to a multiclass classification. The `OneVersusAllClassifier` and `OneVersusOneClassifier` classes

provide implementations to perform a multiclass classification using respectively M and $\frac{M(M-1)}{2}$ binary classifiers.

Regression and classification algorithms implemented in ORION include:

- `KNN`: k nearest neighbors algorithm (classification and regression)
- `QuinlanC4_5`: C4.5 algorithm proposed by Quinlan as an extension of his ID3 to handle continous attributes. This algorithm train decision trees (classification)
- `SVM`: Support Vector Machine algorithm (classification and regression) implemented in ORION using libSVM (Chang and Lin, 2011)
- `FFNN`: Feed Forward Neural Network trained by the back propagation algorithm. (classification and regression)
- `IOHMM`: Input-Output Hidden Markov Model trained by Baum-Welch algorithm (classification)
- `NaiveBayesNet`: classification using a Bayesian inference engine
- `AdaBoost`: classification using multiple weak classifiers.

This list can be extended by implementing the `Classification` or `Regressor` interface. These implementations are then available in our tool via the reflection mechanism of the used language (here, Java).

3.3 ORION Behavioral Model

ORION's behavioral model is an extension of behavior trees. Section 3.3.1 presents the concepts underlying this model and in section 3.3.2 we present the extensions of this model made in ORION.

3.3.1 Standard Behavior Trees

The BT model is the structured architecture which, as we have seen in section 3.1, seems to best match our requirements. We saw that the FSM and BT are the most used in the game development architectures. The FSM allow to simplify the development of AI, but they have the following shortcomings:

- Transitions from one state to another must be explicit, resulting in excessive complexity when the number of states is large.
- A state machine is not a Turing machine. As a consequence, some behaviors are impossible to describe with FSM.

- They are not goal oriented. FSM are designed to be event oriented: events are needed to cause state changes. So it is quite difficult to implement goals with priority management among them for example.
- They are not adapted to parallelism.

BTs do not have these problems. For this reason, we believe that BTs seem better suited to the development of an AI solution in video games.

As the name suggests, the BT model is represented by a tree structure. Each node of the tree represents a goal. Each child of a node represents a sub goal. When the nodes are executed, it returns their current status. This state can be:

- `Success`: Execution succeeded
- `Fail`: Execution failed
- `Running`: Execution in progress

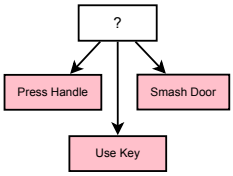
In its most simple implementations, at each time step, the tree is traversed from the root node spreading deep into the branches to the leaves. We study in more detail the different types of nodes in sections 3.3.1.1, 3.3.1.2 and 3.3.1.3.

3.3.1.1 Leaf Nodes

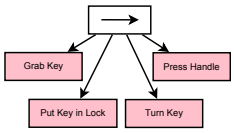
The leaves of the trees are mainly of two types: condition and action. The condition nodes return `Success` if the condition evaluates to true and `Fail` otherwise. They are used to condition the execution of actions or goals. For example, a node describing the behavior of attacking an enemy will be preceded in the tree by a condition node *Enemy present?*. Action nodes perform operations on either the agent environment or its internal state. Possible actions are varied and depend largely on the kind of agent implemented by BT. For example, a node can add data to the internal state of the agent, play an animation, send a command to a motor, write to a file, etc.

3.3.1.2 Composite Nodes

Composite nodes are used to control the execution of their child nodes. There have two main types: Sequence and Selector.



A Selector node (**?**) executes its child nodes **until one of them** returns `Success`. This type of nodes allows to reach a goal by testing several solutions. Thus, in the example on the left, in order to open a door, the agent will try to press the handle, if that fails it will to use a key, and in the end, if the two previous solutions failed, it will break the door down. The selector will return `Success` if one of its child nodes succeeds and `Fail` otherwise. This node allows to manage prioritized actions required to achieve a goal.



A Sequence node (→) executes its child nodes **until they all** return `Success`. This type of nodes, thus, allows to perform an action sequence in order to achieve a goal. In the example on the left, when trying to open a door, the agent will successively try to get the key, and if successful, try to insert it in the lock, if successful it will try to turn the key, and finally, if all the previous actions are successful, it will press the handle. The node will return `Fail` if one of its child nodes fails and `Success` if all succeed.

Other types of composite nodes exist, such as parallel nodes, executing its child nodes simultaneously, random nodes, running a child chosen at random, etc.

3.3.1.3 Decorator Nodes

Decorator nodes have only one child node. These nodes are used to add execution control features. Most common decorators are, for example:

- Repeater: Executes its child node a determined (passed as parameter) number of times. Many variations are possible, as RepeatUntilFail repeating the execution until the child node fails or a specified condition becomes false.
- Succeder: Executes its child nodes and returns `Success` whenever the child node succeeded of failed.
- Failer: Executes its child nodes and returns `Fail` whenever the child node succeeded of failed.
- Limiter: Executes its child node but limits the maximum number of ticks. Thus, a node can return `Running` a maximum number of times. If the execution is not completed after this number of tick is reached, the node returns `Fail`.

3.3.2 ORION Behavior Tree Extensions

We have seen how the BT is endowed with many qualities in order to achieve an AI in video games. So, we choose to extend this model. First, without losing the potential of BT to add *ad hoc* code features, we use as an execution context of BT, a `VariableSet`. This context contains, at each tick, data collected by the agent (its perceptions) and allows the different nodes to change this context (add or change data). The data being typed, each behavior can control its consistency and performs introspection on these data. It would further be possible to control, during the design process of BT, that the data used by each behavior will be present in the context before their executions. However, this is not implemented in ORION. We now propose to extend BT to be able to use the data mining techniques we studied in chapter 2. We presented, in section 3.2, a model encompassing the various tasks achievable by data mining techniques. Here we present the integration of the structural model into the behavioral model.

3.3.2.1 Data Mining Behaviors

The most important functionality we want to add to BT is the possibility of using behaviors that have been learned from a dataset. To do this, we propose to add node types to the BT model. First, we add a particular type of node composed of a `FunctionLearner`, to predict an output variable based on input variables. So, this is an action node that changes the context by adding the predicted value in it. We also add a new type of composite node to choose which child node to run through the prediction of a classifier. Finally, we add an action node composed of a `DataTransformer`, to transform the data. Figure 3.9 presents our model of Behavior Tree.

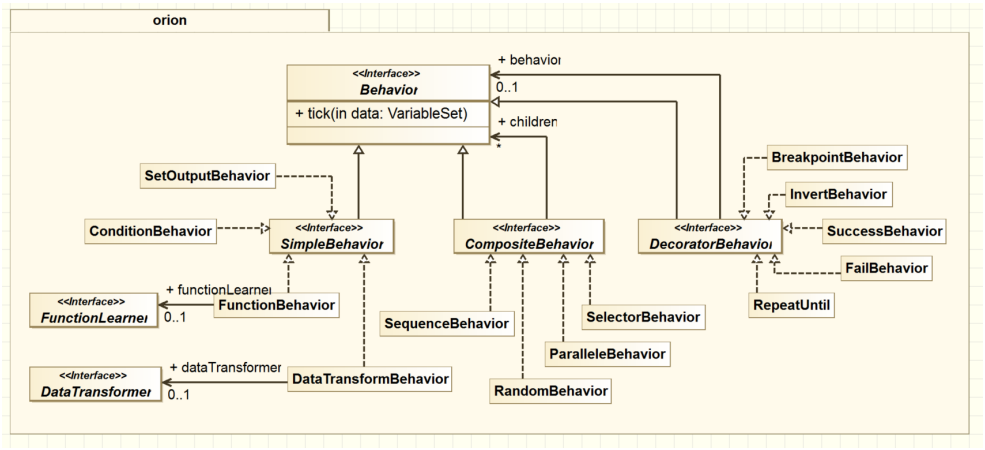


Figure 3.9: ORION data mining behaviors

3.3.2.2 Online and Offline Learning

We want the construction of BT to be possible both online and offline. The offline construction is simply enabled by our implementation of ORION, manually constructing the tree and setting the node parameters (by associating a `FunctionLearner` previously trained with data to a `FunctionBehavior` for example). But online training requires integrating other mechanisms. Indeed, many additional problems arise when trying to learn online. Most data mining techniques we studied in the previous chapters are not iterative. It is not possible to start learning each tick. We must therefore offer a flexible method to trigger a learning algorithm, following an event in the game or after a number of tick for example. BT's formalism allows to simply achieve this kind of behavior: we decide to add to our other models some node types that will collect data and perform learning on the data collected.

We propose the following nodes:

- `StoreDataBehavior`: Add data to a `Dataset`
- `TrainBehavior`: Train a `FunctionLearner` with a `Dataset`
- `DataTransformBehavior`: Perform data transformation on a `Dataset`

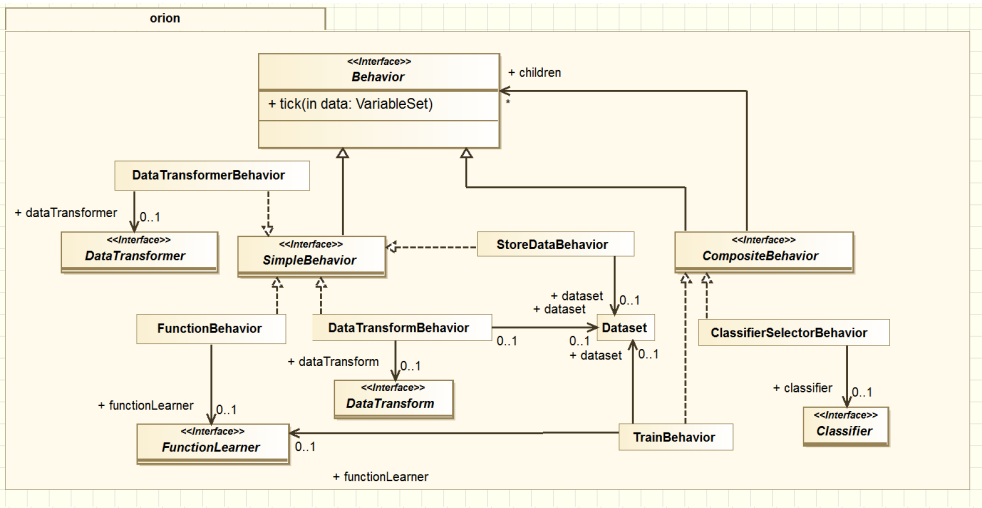


Figure 3.10: ORION online learning model

Figure 3.10 illustrates the implementation of these features in the ORION model.

The typical use of these units in order to perform online learning is shown in figure 3.11. In this example, the left side of the tree allows the execution

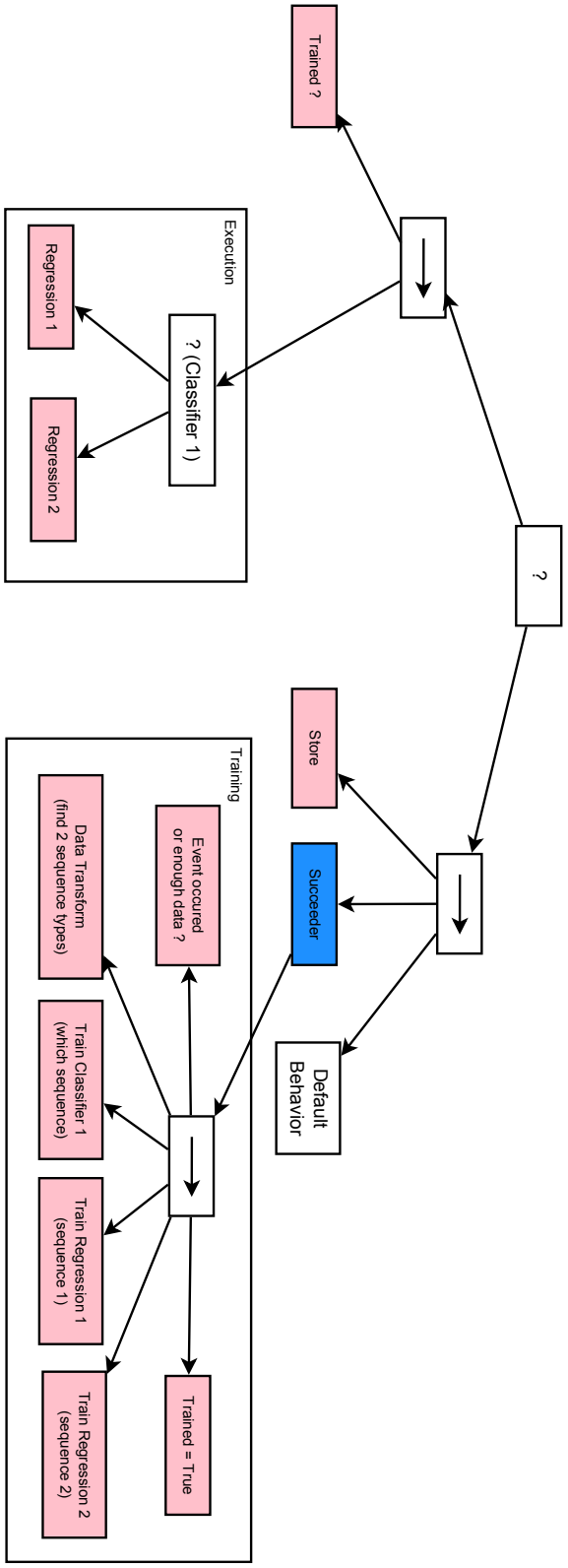


Figure 3.11: Typical use of ORION Behavior Trees for online learning

of the behavior performed once learning is done and the right part allows directed learning and implement a default behavior if learning has not been achieved. Thus, if learning has not yet been done, the condition node on the left (*Trained?* in the figure) returns `Fail` and the right node is executed. The current context is then stored in a `Dataset`. If the number of data in the `Dataset` is sufficient or an event occurs in the game for example, learning is carried out in several steps. First, the `Dataset` is divided into $\boldsymbol{n}$ sequences by a data processing algorithm (usually a time series clustering algorithm). These sequences belong in this example to two different types. Then, the three `FunctionLearner` nodes associated both with the execution part (the three nodes references *Classifier 1*, *Regression 1* and *Regression 2* in the figure) and the Train nodes are trained with the `Dataset`. Finally, the execution context is changed to indicate that learning was achieved (Trained = True). The following tick therefore executes the train behavior (left side of the tree). As long as learning is not finished, the default behavior (implemented with *ad hoc* features for example) is executed.

3.4 Conclusion

In this chapter, first of all, we justified the use of a structured general architecture to enable the design of AI in video games. These architectures allow indeed a flexible design and improve the robustness, maintainability and testability of the solution. Moreover, they allow better efficiency in teamwork.

Then, we presented the ORION model. ORION permits to associate semantics to the data. This semantics is essential to enable a better understanding of the results of data mining algorithms we want to use. ORION also provides a powerful data visualization solution. Traditional charts for univariate or multivariate analysis of the data set (histogram, radar and parallel plot, etc.) are implemented and can be extended. A very generic representation mode allows the data to be displayed in 2 or 3 dimensions. Data visualization primitives such as points, vectors, images, texts or graphs are proposed and the model allows extensions. This allows us to display, in a generic tool, most of the data available on the UCI database.

The ORION model formalizes various techniques coming from research in data mining and proposes to gather them together according to their use. Our tool, therefore, proposes to perform tasks as diverse as data clustering, vector quantization, extraction of characteristics or prediction. The implementation of our model provides a generic approach for data analysis and data mining in the manner of WEKA or ELKI. The main contributions of our tool in relation to the latter are the management of semantics and the greater possibility of

visualization. The number of available algorithms is smaller than the ones provided with WEKA but may easily be extended.

Finally, ORION also offers a behavioral model. Since we have decided that our solution should be based on a structured general architecture, we chose to extend the BT model for our needs. So, we add to this model the ability to use data mining techniques to implement complex behaviors. This model allows both online and offline learning.

Contents of Chapter 4

Chapter 4

Applications: Pong and Unreal Tournament 3

> I am putting myself to the fullest possible use, which is all I think that any conscious entity can ever hope to do.
>
> HAL - 2001, A space Odyssey (1968)

Summary

In this chapter, we show two applications of our Orion *model. We start by applying data mining techniques to the video game Pong. We analyze the data from game sequences played by human players. We show that the exploratory analysis allows us to identify various strategies implemented by players. We also identify the evolution of these strategies during gameplay and some invariant behavior between players.*

We continue using unsupervised analysis methods to segment our data into homogeneous groups. We try to identify the player strategy corresponding to a given situation. We will see that we are, to some extent, able to perform this clustering automatically. However, we evoke reservations about the applicability of this method.

Then we use and evaluate supervised learning techniques to reproduce part of the behavior. To do this, we first annotated game sequences with a semi-automatic method using rules. Then, each identically annotated sequence is used as training data for the regression algorithms tested. We evaluate the performance of each algorithm. We then use a classification algorithm in order to choose the most appropriate low level action to execute, producing the overall behavior.

Subsequently, we use Orion *to implement a bot for the game Unreal Tournament 3 (UT3). We implement the* Chameleon *model (Tencé, 2011) with* Orion*. Then, we propose an improvement of the* Chameleon *method in order to learn the environment automatically. Still using our model, we use that improvement to make the bot movements more in line with those of a human player. We also learn aiming and close combat behaviors through supervised learning techniques.*

Summary

Finally, we implement the overall behavior using the ORION *BT, integrating ad hoc features in order to manage aspects of behavior that we have not been able to learn automatically.*

Introduction

In the previous chapter we have presented our model, ORION. We built this model in order to implement artificial intelligence in a video game. Our ambition is to learn behaviors by analyzing data from real players. Some commercial games offer this type of learning like for instance *Forza Motors 5*, which attempts to replicate how a player drives in order to propose its digital clone. The techniques used in this game are not easily transferable to other games and results are still fat from perfect. This is a good indication of how challenging this task is. That is why we use our model in a somewhat more pragmatic manner since we built a *Pong* and an UT3 agent with ORION using learning methods but still exploit expert knowledge.

Plan

The section 4.1 of this chapter focuses on using ORION to achieve artificial intelligence in the context of the *Pong* game. In section 4.1.2, we perform an exploratory analysis on data collected with our implementation of the game. This analysis allows us to discover some invariant in players' behaviors. In section 4.1.3, we use clustering techniques to automatically identify the strategy of the players. These methods do not, however, give us full satisfaction regarding the results we get. Therefore we apply an expert rule method in order to achieve this identification. In section 4.1.4, we use regression algorithms to reproduce the behaviors identified by a semi-automatic method in the previous section. The regression algorithms are tested with different parameters, allowing us to compare their performances. In section 4.1.5, we implement the overall behavior using ORION's BT with both offline and online learning.

Section 4.2 describes the use of ORION in the design of a Bot for the game Unreal Tournament 3 (UT3). We start by describing the data collection process in section 4.2.1. We then, in section 4.2.2, show how we use ORION in order to visualize and transform the collected data. In section 4.2.3, we use ORION to implement the CHAMELEON model proposed by Tencé (2011). We then, in section 4.2.4, show how we learn low level behaviors. Finally, section 4.2.5 shows how we implement features of the Bot behavior which we have not been able to learn automatically.

4.1 Pong

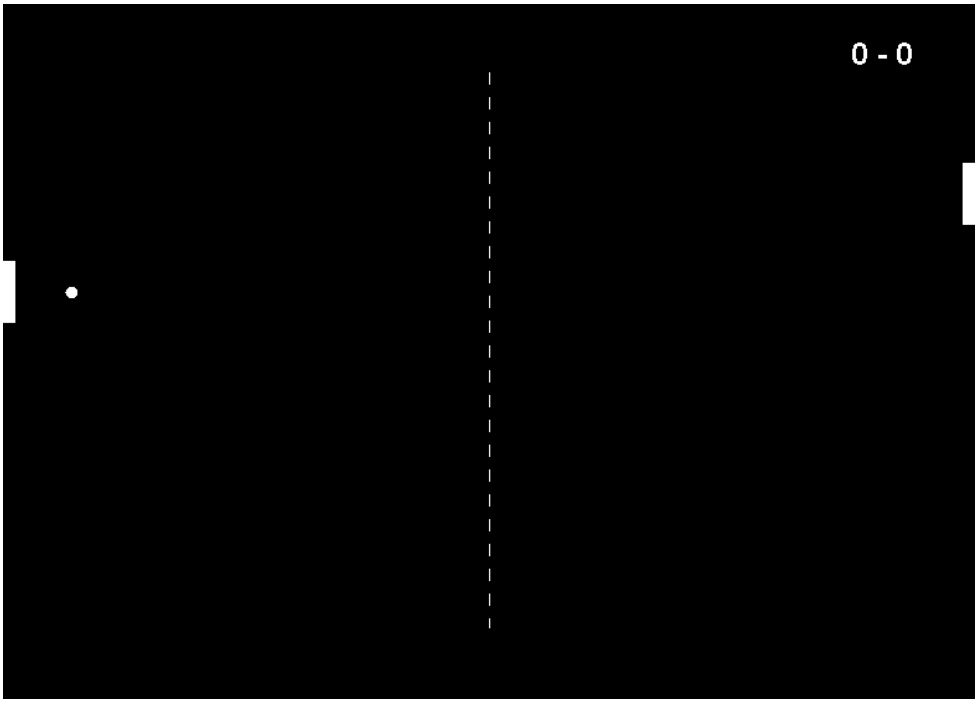


Figure 4.1: *Pong* screen

The game *Pong* was one of the first video games to enjoy popular success. This is a table tennis game where the ball passes through the screen laterally. Players vertically control a paddle on one of the edges of the screen and try to make the ball bounce on it. A point is scored by the opponent when a player fails to catch the ball with the paddle before the ball touches the edge of the screen. Figure 4.1 shows the screen viewed by players. In our implementation, a frictional force is applied to the ball when the paddle velocity at impact with the ball is not zero, thereby deflecting its trajectory. The ball speed (the norm of the vector) is constant. The player controls the paddle with a mouse. When a point is scored, the ball is placed at the center of the playing area and the player who lost the point must press a mouse button to put the ball into play. The ball is then launched with a velocity vector defined randomly.

4.1.1 Data Collection

We asked five persons to play in our implementation of the game and collected the traces: five traces when the players face a bot following the ball perfectly, that is to say it can not lose, and five traces when human players play against each other. The data collected are:

- position of each paddle
- ball position
- state of the mouse buttons
- current scores

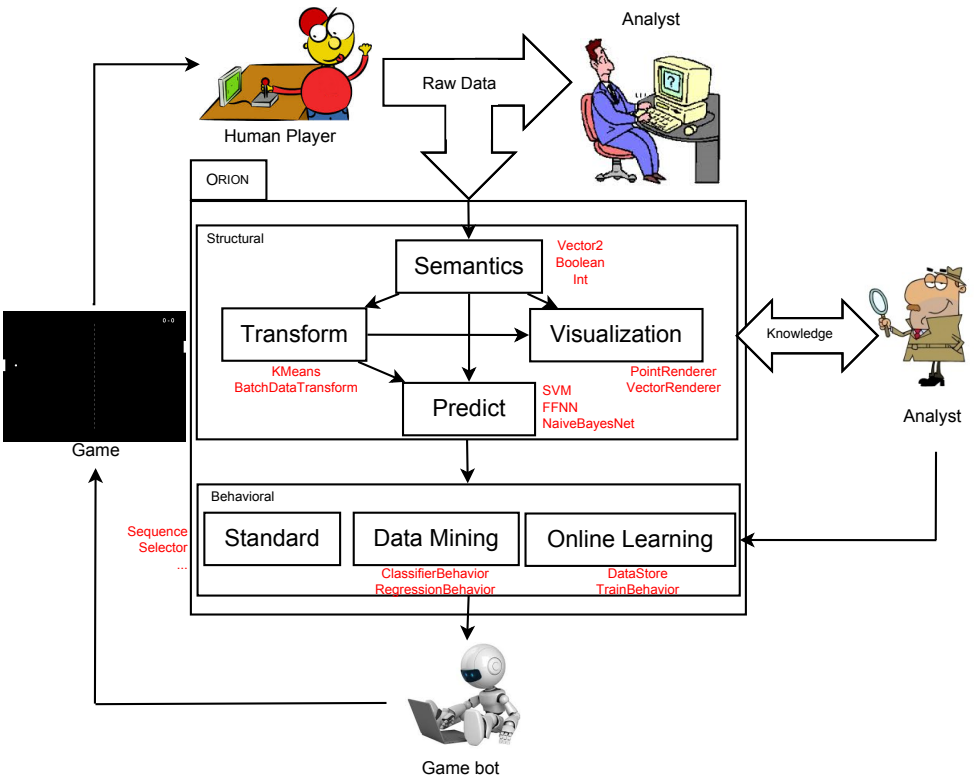


Figure 4.2: ORION Workflow for *Pong*

4.1.2 Explanatory Analysis

In this section, we perform an explanatory analysis of the collected data.

4.1.2.1 Add Semantic

Exploratory analysis is used to retrieve valuable information from the data. In order to illustrate this concept, we study the previous traces. First, to conduct this analysis, we need to find representations of our data that give the analyst the best understanding. In the case of *Pong*, it may be game reconstruction because each datum in the traces of the game has an understandable semantics. The ball position is a two-dimensional vector (`Vector2`, see 3.2.1

page 50), paddle positions also (although one of the coordinates of each paddle is fixed, because the horizontal movement of the paddle is impossible). The scores are simple integers (`Int`). The state of mouse buttons is boolean (`FiniteSet`).

4.1.2.2 Data Visualization

ORION offers the aforementioned types and the visualizers of these semantics. Therefore, we can simply reconstruct the game from the traces. To do this, we use three `PointRenderer`s (see 3.2.3 page 55) to represent the ball and two paddles, and two `TextRenderer`s to view the scores. The `PointRenderer`s used for paddle representation are configured so that their color correspond to the state of the mouse buttons. Thus, the dataset can be displayed with a relevant representation with respect to its semantics. Figure 4.3 illustrates reconstruction of a game state within ORION.

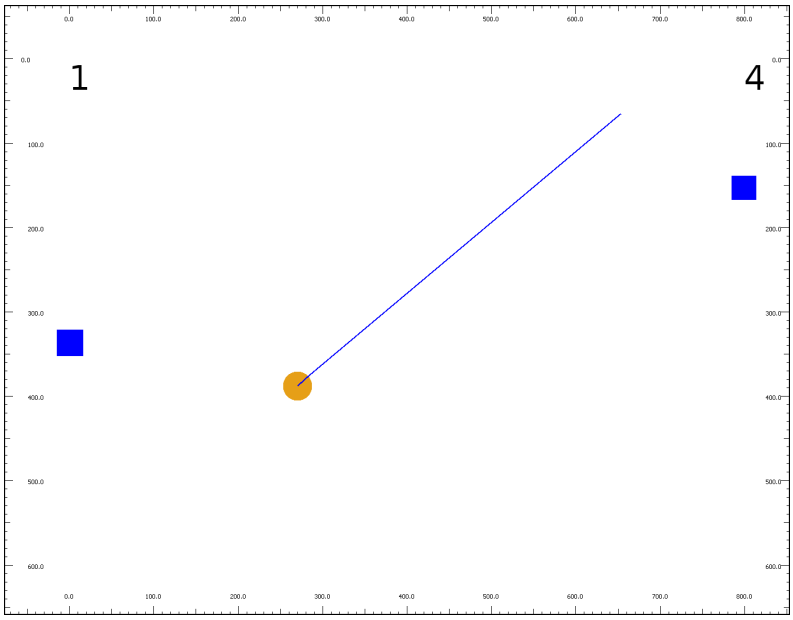


Figure 4.3: *Pong* game visualization in ORION

In our case, the traces we are studying are time series. Therefore it is appropriate to use adapted visualization methods, such as the evolution of data over time. Figure 4.4 illustrates the representation from ORION of a few seconds of a game. The top diagram shows time evolution of the player's paddle position (red curve) and the vertical position of the ball (green curve). In addition to finding a high correlation (which seems logical enough because the player's goal is to catch the ball), it can be observed that, when the ball touches the paddle, the position of the player's racket tends to evolve faster.

This illustrates that the player tries to use the friction force to deflect the ball and give it a more difficult trajectory for the opponent to catch the ball. The bottom curve shows the development of the paddle positions of each player (red curve for the left player and blue curve for the right one) when the ball is stationary. We can see that a kind of "question and answer" game sets up between the two players. When one oscillates the position of his paddle, the second does the same.

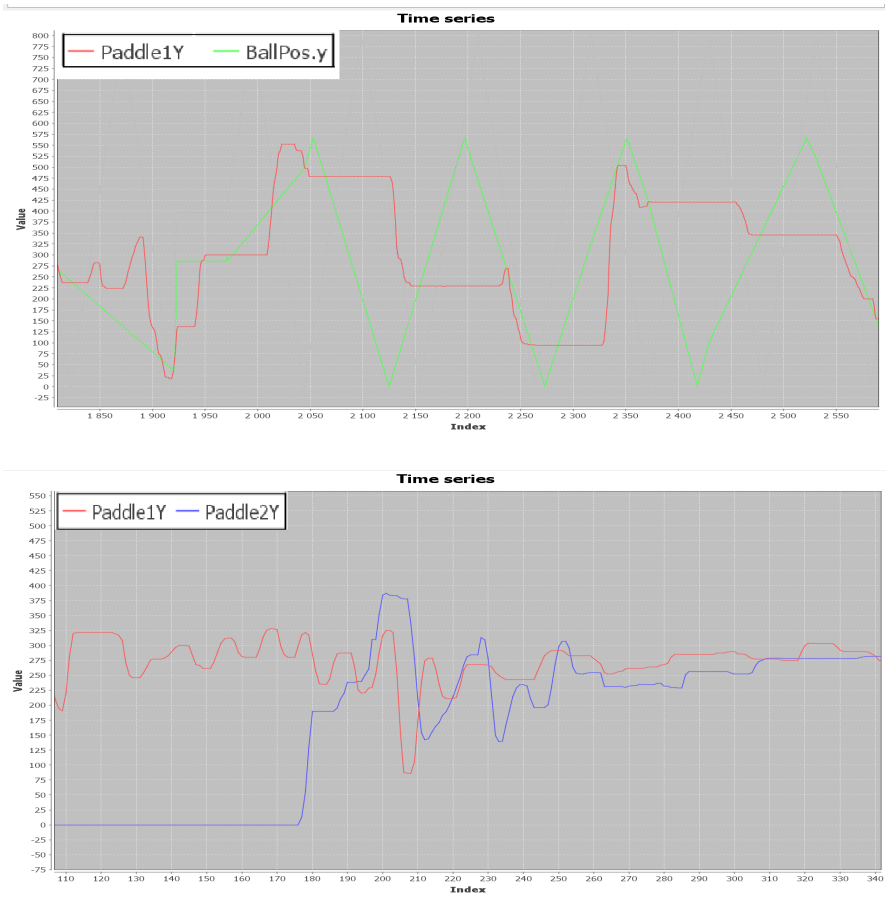


Figure 4.4: Time series visualization in ORION. Top: Interaction between the player and the ball. Bottom: Interaction between the two players when the ball is not moving

Still using our visualization model, we are able to represent the data in other ways. For example, in figure 4.5, the horizontal axis represents the vertical position of the ball and the vertical axis represents the vertical position of the player's paddle. The color represents the horizontal position of the ball. Thus, the more blue the point's color is, the more the ball's distance to the

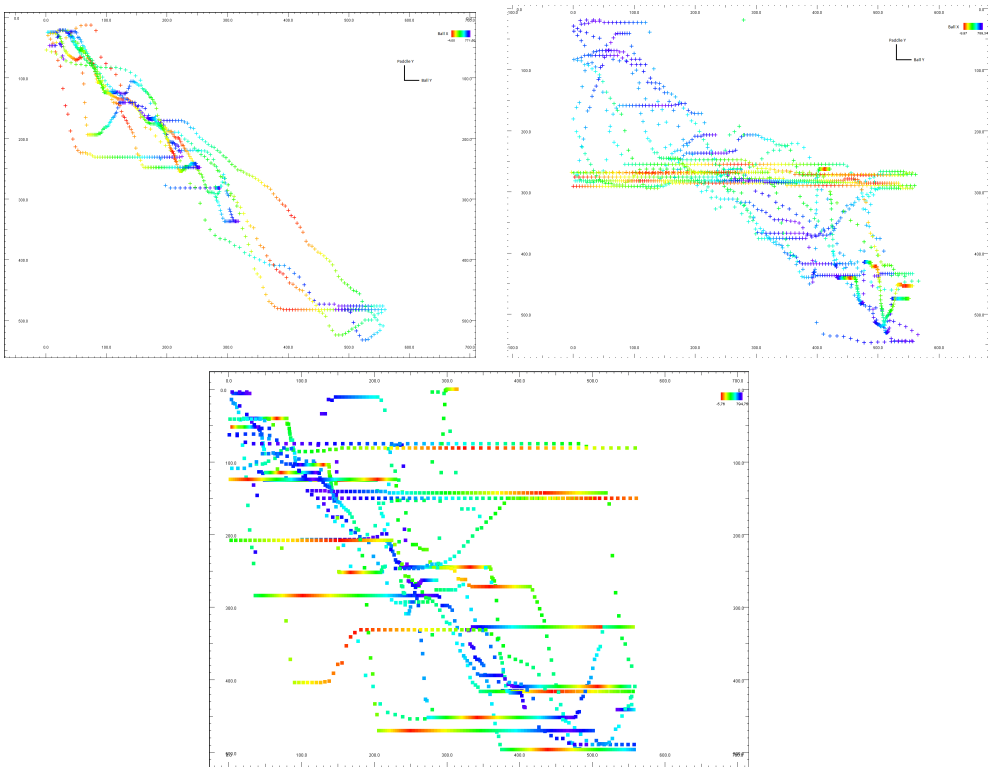


Figure 4.5: In the 3 diagrams, the horizontal axis represents the vertical position of the ball, the vertical axis represents vertical position of the paddle and color represents the horizontal position of the ball

edge decreases. This diagram shows three different sequences. On the three charts blue points are very close from the diagonal. This is not surprising since it expresses that when the ball is near the edge defended by the player, the vertical positions of the paddle and the ball are similar. The players are simply trying to catch the ball on the three charts. However, we note that in the first chart, all points are approximately on the diagonal: the player was applying the basic strategy of always following the movement of the ball. On the second chart, a majority of non blue points stand on a horizontal line one the center of the screen. Therefore we can detect that the player was using a different strategy: let the paddle in the center of the screen and move towards the ball only when it approaches and then re-position the paddle at the center. Finally, in the third chart, the horizontal lines are present over the entire height of the screen. The player's strategy was therefore to only move when the ball came closer and stay still when the ball is not close. So we identify three different strategies used by players. Moreover, these strategies can be used in succession by the same player in the same game.

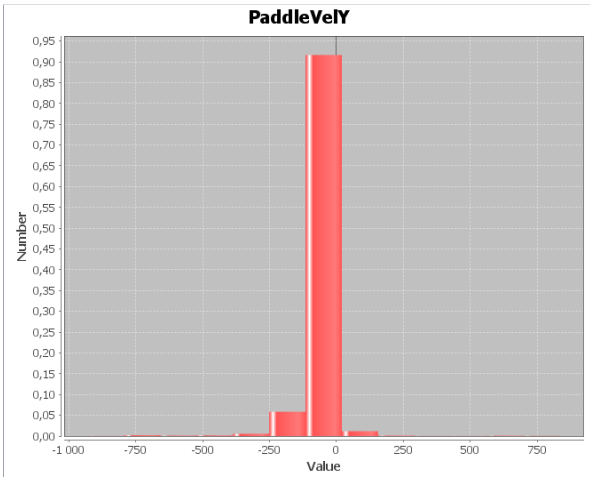


Figure 4.6: Paddle speed histogram

4.1.2.3 Transform Data

Exploratory analysis is not based solely on data visualization. Other processes can help in this analysis. Dimensional reduction, studied in chapter 2, is notably one of the most useful techniques to perform exploratory analysis. Here, however, this method does not seem relevant as the number of dimensions is relatively small and the dataset is easily displayable through its semantics. A common data transformation for time series consists in computing the derivatives of some attributes over time. Applied to the positions of the ball and paddles, we get their speeds. This technique can, for example, be sufficient to achieve regression or classification on time series with algorithms not being specially dedicated. This transformation in ORION is implemented by the `Differentiate` class (see 3.2.2 page 52). In addition, it provides additional information to the analyst that can lead to relevant conclusions. Here, we found an invariant between players with a simple univariate analysis. Figure 4.6 shows the histogram of the paddles speeds. We find a similar histogram for all our players: indeed it seems that players are likely to apply friction to the ball by pushing the mouse rather than pulling it. Without alleging any reason of this invariant (which is perhaps due to a bias in our data collection or a coincidence given the low number of players), it is interesting that the implementation our AI reproduces it. This process illustrates one of the main task performed by exploratory analysis: exploring the data without any *a priori* question and discovering knowledge that was not expected beforehand.

4.1.2.4 Conclusion

In conclusion, on the example of the game *Pong*, exploratory analysis of human traces allowed us to:

- Reconstruct the different game sequences
- Identify characteristics of a low-level behavior (a frictional force applied to the ball)
- Identify different strategies used by players (follow the ball, back to center, imitated the opponent, etc)
- Identify an invariant between the players (the friction force applied in one direction rather than in another)

These findings constitute knowledge that can be implemented by an expert to build a believable behavior. It can also be used to accurately select learning algorithms to be used to reproduce these behaviors. Finally, it may help to validate the learned behaviors *a posteriori*.

4.1.3 Unsupervised Analysis

Unsupervised analysis, including data clustering, is an effective way of performing a discretization of continuous variables. This discretization may be necessary if one wants to use learning algorithms that accept only discrete data as an input (such as Quinlan (1986)'s ID3 or most implementations of Bayesian networks). Since we want to use such algorithms (see section 4.1.4.2), we have to discretize our data. If we want to perform this discretization automatically rather than arbitrarily choosing the limits of each discrete value, we may use a clustering algorithm. Such an algorithm would divide our data into a number of homogeneous groups, thereby performing discretization. However, clustering algorithms usually need the whole set of data but, in our case, once the clustering is achieved, we would like to be able to place any new datum in the more appropriate cluster. This may be a problem, for instance, in the case of density-based algorithms such as DBSCAN, because processing new data once the clustering is done is impossible. A simple solution would be, once the clustering operation done, to perform discretization as a classification problem. But since it can be time consuming, centroid based clustering algorithms seem a better approach. With these algorithms, clusters are defined as the Voronoi cell of each centroid. Knowing the centroids characteristics, one can find the cluster of new data by finding the closest centroid. In ORION, we provided two interfaces that distinguish, among algorithms, those that are able to transform a single datum and those which need all dataset: `DataTransform` and `DataTransformer`. A `DataTransformer` is a `DataTransform` able to perform a transformation on a single datum. The only data clustering algorithm implemented in ORION and implementing the `DataTransformer` interface is the `KMeans` class. Therefore, by default we will use this algorithm to perform a discretization.

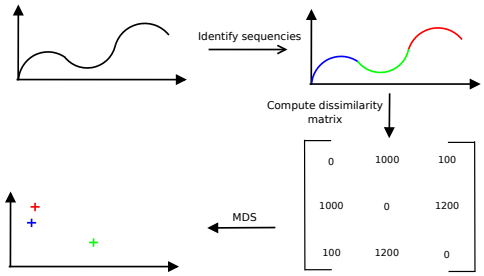


Figure 4.7: Time series conversion process

4.1.3.1 Time Series Segmentation

Clustering can also be applied to time series: in this situation, the goal is to segment the time series and annotate each segment. Thus, two segments annotated in the same way are considered to be close. In the case of *Pong*, the idea is to use this technique to automatically find the strategy employed by the player. To test this approach, we asked a player to alternately use one of three strategies that we had previously identified in our data. We did not impose any particular order nor particular constraint on the distribution of these three strategies. The purpose of this experiment was to check if we were able to correctly segment a sequence that does not result of synthetic data. However, those data need to be sufficiently clean so as not to have to deal with various problems like excessive noise.

First, our segmentation process splits the dataset into roundtrip sequences of the ball. Then we compute the distance between each sequence using a Dynamic Time Warping (DTW) distance. Informally, the DTW distance consists in finding an optimal alignment of data between the two time series and computing the sum of distances between each datum aligned identically in the series. The distance problem between the time series and the DTW are explained in appendix A.4. Once calculated, the distance matrix enables a multi dimensional scaling (see chapter 2 section 2.2.2.2). Each rountrip sequence is then represented by a single datum. The applied process is shown in figure 4.7.

Once this process is achieved, standard clustering becomes possible in order to gather the most similar rountrips. By performing this process on our dataset (which could be described as semi-synthetic), the result of the time series clustering was relatively satisfactory, but this process needs many parameters to be set. Clustering (the final step) has been produced with the KMean algorithm, and we set $\boldsymbol{k = 3}$ since we wanted to identify the strategies applied by the players identified in section 4.1.2 (there were three strategies). In addition, the DTW itself uses a distance we also had to choose. Again, we tested different distances and selected the one that made the result more

	Ball moving	
	Opponent moving	**Opponent fixed**
Player moving	*Catching* if ball is coming close *Reposition* if ball has just left *Imitate* otherwise	*Catching* if ball is coming close *Reposition* if ball has just left *UnknownMove* otherwise
Player fixed	Wait	Wait
	Ball fixed	
	Opponent moving	**Opponent fixed**
Player moving	*Imitate*	*UnknownMove*
Player fixed	*Wait*	*Wait*

Figure 4.8: Rule based segmentation

in line with our expectations. It is therefore no surprise that by testing the same approach on a set of real data, we did not obtain satisfactory results. Nevertheless, this approach to time series segmentation seems interesting and, in our view, deserves to be thorough.

Since the previous method does not allow to achieve a satisfactory time series clustering of our data, we have implemented a much more pragmatic approach by exploiting the knowledge we had extracted during the exploratory analysis. We identified that players perform the following low-level actions:

- *Imitate*: This is the game of questions and answers between two players when the ball is stationary
- *Catching*: Get the ball
- *Reposition*: Move the paddle at the center of the screen
- *Wait*: Wait for the ball to get close

These low-level actions can be considered as primitives to reproduce the strategies identified in section 4.1.2. Also, in the different dataset of real games, many segments can not be identified as one of these low level behavior. Therefore we need to add another low level behavior to handle this as noise (*UnknownMove*).

Thus, we implemented a fairly simple algorithm, design to identify these behaviors in a time series with simple rules. These rules are presented in figure 4.8.

	FFNN		KNN		SVM	
	MSE	**Time**	**MSE**	**Time**	**MSE**	**Time**
Imitate	2879	48s	3598	N/A	990	<1s
Catching	3292	122s	6640	N/A	2532	5s
Reposition	6016	46s	8924	N/A	1817	<1s
UnknownMove	3974	32s	5239	N/A	3653	<1s

Figure 4.9: Regression of low level behavior results

4.1.4 Data Prediction

4.1.4.1 Low Level Behavior Reproduction

In section 4.1.3.1, we achieved the low level behavior identification through expert defined rules. Now we need to replicate these behaviors individually. This is a typical use of regression algorithms. We tried regression on these low level behaviors using the following algorithms: Feed Forward Neural Network (FFNN), KNN and SVM. Table 4.9 shows the results. The algorithm parameters were investigated by cross validation and grid search. We used KNN with the Euclidean, Manhattan and Mahalanobis distances (see appendix A). Neural networks have a topology with one or two hidden layers having a number of neurons varying from 3 to 25. We used Mean Squared Error (MSE) to compute error in the results.

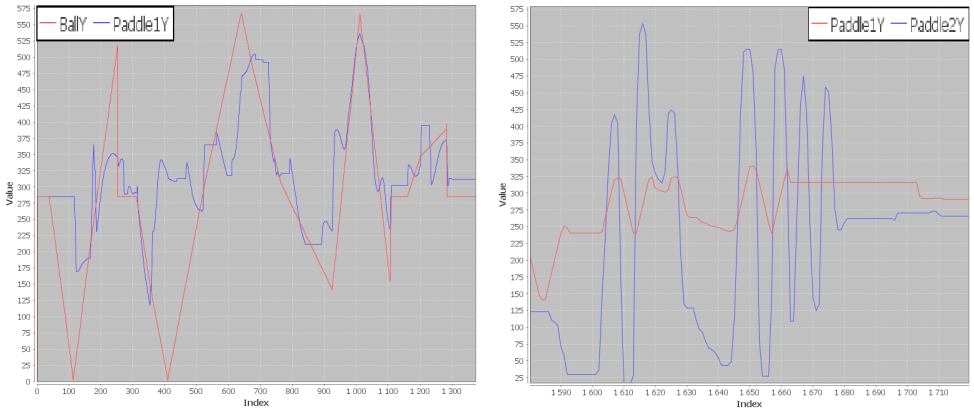


Figure 4.10: Time series visualization of learned low level behaviors. Left: Catching behavior. Right: Imitating behavior

While MSE provides quantitative information about the learned behaviors, qualitative analysis of those behaviors may also be conducted. Figure 4.10 shows the resulting behaviors. We can see that the two curves look like

those shown in figure 4.4, that display real human data. However, our feeling while looking at the result in the game is that the Imitation behavior lacks amplitude in its imitation movements. This is not surprising because the questions-answers game is a chronological process and we use regular regression algorithms that do not fit time series analysis. Moreover, movements of the Catching behavior seem a little jerky sometimes. The overall results are, nevertheless, quite satisfactory from our point of view.

4.1.4.2 Strategic Level Design

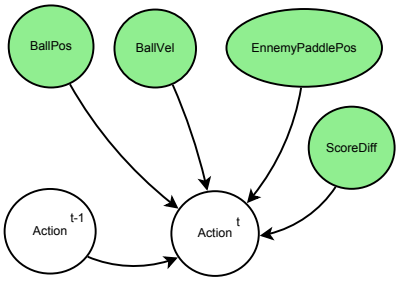


Figure 4.11: Naive Bayesian Network Classifier for *Pong* action choice

As already mentioned, the low level behaviors identified in section 4.1.3.1 and reproduced in section 4.1.4.1 can be seen as primitive behaviors of our agent. Thereby, we have to use these primitives to produce an action sequence that is similar to those observed in our data set. To do this, we use a very simple naive Bayesian network, as shown in figure 4.11. In this model, low level actions at a given tick depends on the ball's position and velocity, the opponent's position, the score difference and the previous action. Since in this model all variables can be observed, learning probabilities is straightforward.

4.1.5 Behavioral Model

4.1.5.1 Pong BT

Now that we have presented all the steps we performed, in order to reproduce the behavior that was partially learned from the data, we need to put all these behaviors together. The BT producing the complete behavior is presented in the not pretty nor ergonomic but functional ORION agent model editor GUI in figure 4.12. The model consists in a sequence containing `DataTransformerBehavior`s linked to a KMeans algorithm in order to discretize data. Then, a `SelectorClassifierBehavior` linked with the Bayesian network is in charge of selecting the low level behavior to execute. The low level behaviors are represented in the BT by `RegressionBehavior`, linked to

the regression algorithm used to learn them. The Wait behavior is implemented by a `SetOutputBehavior` (set paddle position to the previous one).

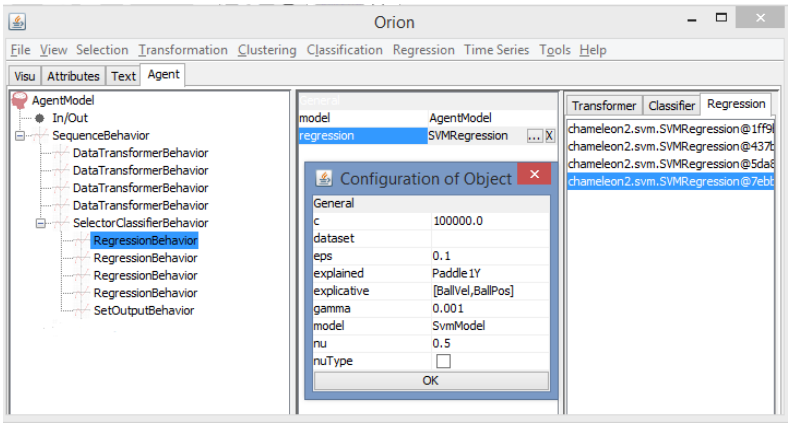


Figure 4.12: ORION Agent Model Editor Displaying *Pong* BT

4.1.5.2 Online Learning

We have added the ability to learn behavior online. For that purpose, we used the method we have already presented in the previous chapter. The BT is presented figure 4.13. In this figure, the left side corresponds to the BT presented in section 4.1.5. The right side shows the management of online learning. The data are stored on each tick (the "Store in Dataset" node). Whenever one thousand data were stored, learning is started. The latter consists in performing discretization of each input data, annotating sequences and training the classifiers and regression algorithms. When no learning has been made, the paddle is simply positioned at the y-position of the ball.

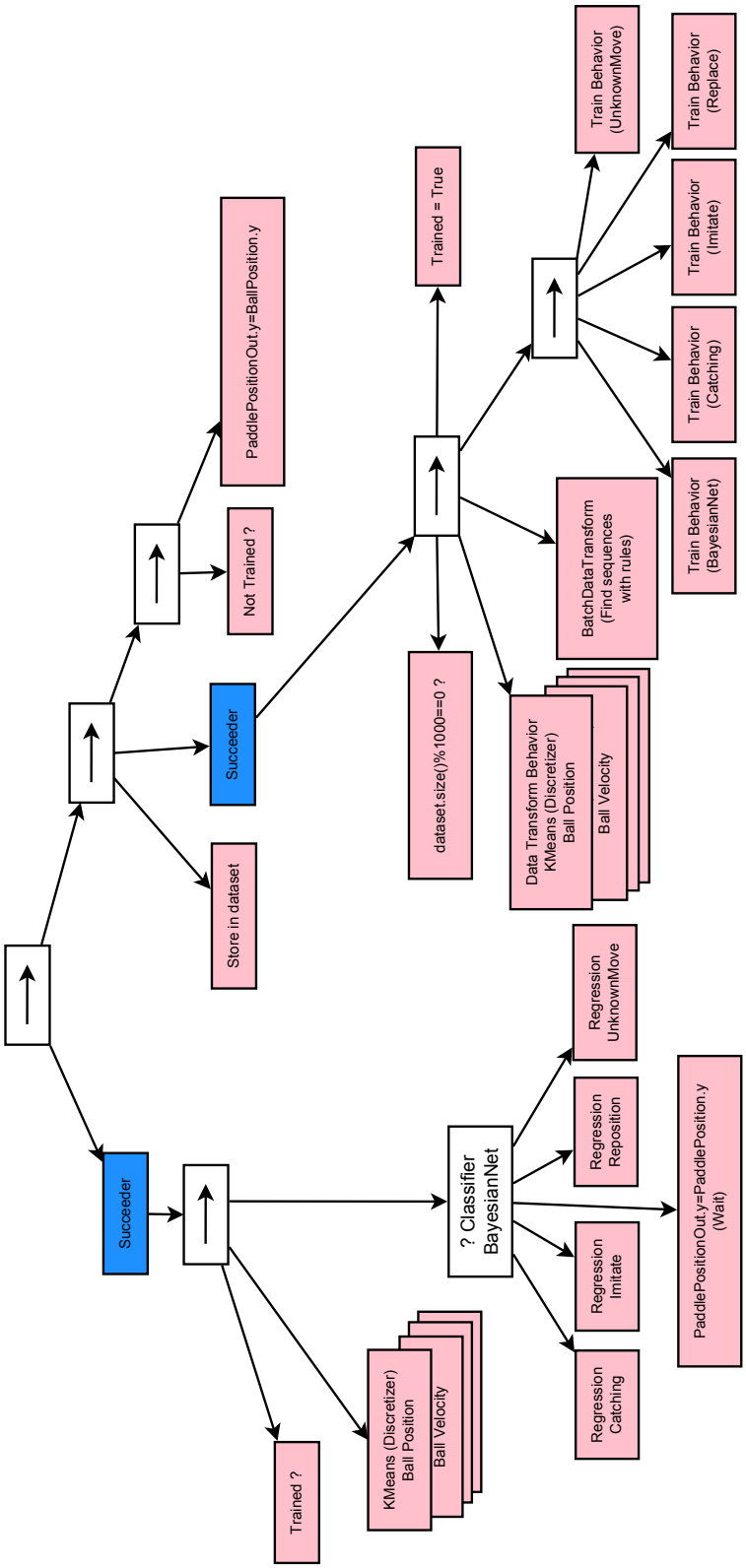


Figure 4.13: ORION BT for Online Learning in *Pong*

4.2 Unreal Tournament 3

Figure 4.14: In-game Screenshot of UT3

Unreal Tournament 3 is a First Person Shooter video game developed by Epic Games and released in 2007. In this game, the player incarnate a character with a variety of futuristic weapons, evolving in an arena with other players. The arena contains items that players can collect. They may be weapons, ammunition, shields, first aid kits etc. Several game modes are available. The simplest one consists of a mode called "deathmatch", in which each player plays against all the other players. Each time the player eliminates another player (by dropping his life points to zero using the weapons at his disposal), he scores one point. The eliminated player respawns (reappears) in another location of the arena. The game is played for a limited time and for a maximum score determined in advance. If a player reaches the maximum score, he wins and the game ends. If the limit time is reached, the player with the higher score wins. Figure 4.14 shows a screenshot of the game.

In order to recover the traces of the players and control the bots in the game, we use the GameBots API. This API, written using the game engine scripting language (UnrealScript) provides a Telnet (very simple network protocol) interface to retrieve the events of the game and control our bot via a separate process of the game, and possibly on another machine. We implemented a Java API, encapsulating a Telnet client that exposes all GameBots features.

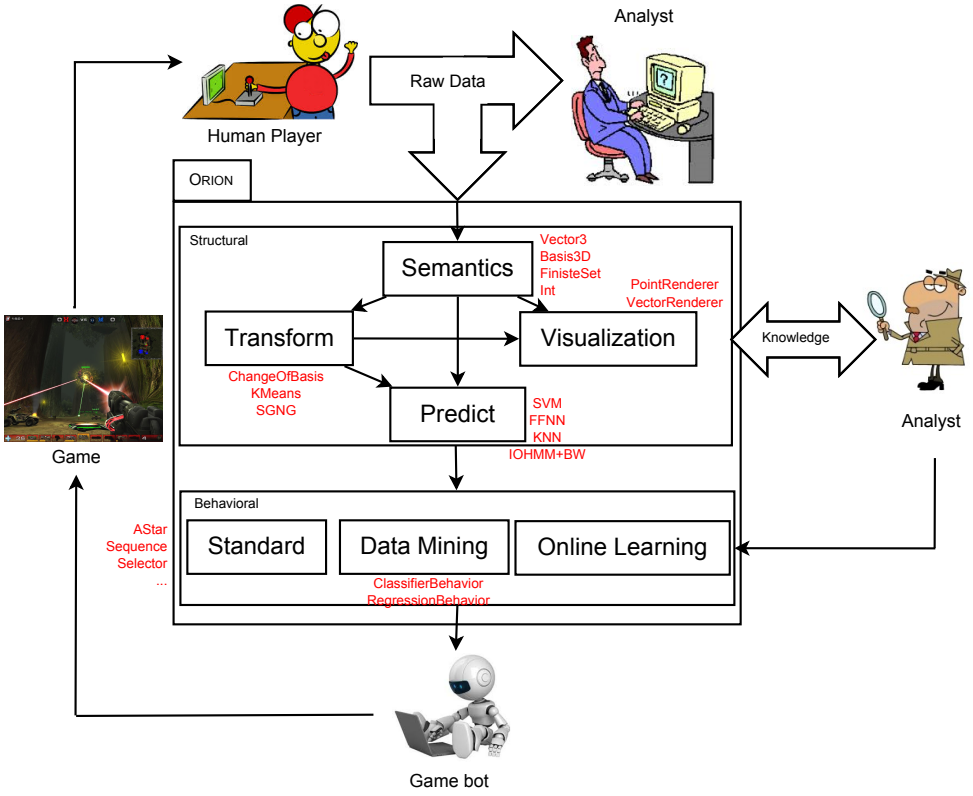


Figure 4.15: ORION Workflow for UT3

4.2.1 Data Collection

4.2.1.1 Raw Data

When observing a player evolving in the game with GameBots, we have access to many information about him and his environment. We choose to collect this unprocessed information in a CSV file. Raw data are processed with ORION as explained later in section 4.2.2. The information collected includes:

- Player position
- Player orientation (pitch and yaw, roll is always 0)
- Player velocity
- Current weapon name
- Life points
- Remaining ammunition

- Shoot (is the player shooting?)
- Number of enemies visible
- Enemy positions (the 3 closest)
- Enemy orientations (the 3 closest)
- Enemy velocities (the 3 closest)
- Number of seen items
- Item positions (the 3 closest)
- Number of seen navigation points
- Navigation point positions (the 3 closest)

4.2.1.2 Low Level Behavior Data Collection

In order to create datasets containing only sequences where a player is performing a specific task, we conducted scripted gaming sessions. A human player performs the tasks defined in our scenarios and game traces are recovered. Three scenarios are proposed:

1. Movement: The player must simply navigate in the environment. No enemy is present. These data will allow us to learn how to move in the environment.

2. Long range aiming: The player cannot move. He has infinite ammunition. He is alone in a large room with a single enemy and only intended to shoot it.

3. Close Combat: The player is in a small room with an enemy and must dodge attacks while firing at the enemy.

4.2.2 Data Transformation and Visualization

Raw data do not allow to easily perform analysis. As ORION adds semantics to the data, we make use of it to facilitate an exploratory analysis. In UT3, most data are spatial locations of elements in the game (players, items, etc). ORION implements several types associated with locations in three-dimensional space and `Renderer`s allowing their visualization. Using these features, we are able to rebuild the game play in ORION as shown in figure 4.16.

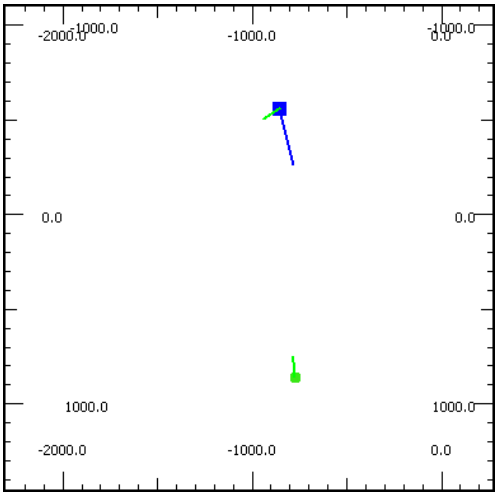


Figure 4.16: UT3 game reconstruction in ORION

Spatial location from the raw data does not allow us to effectively reproduce the observed behavior. The position of the player, for example, should not be used as input data of a learning algorithm. Doing so, the learned behavior would only be able to reproduce actions at the specific locations where they were observed. Therefore, we need to transform all data related to location in the environment to the local coordinates of the player. The `Basis3D` type, implemented in ORION, is used to represent a position and orientation in space. This type is implemented through a matrix of homogeneous coordinates and corresponds, from a mathematical point of view, to a 3D orthonormal basis. Spatial transformations become possible in ORION when data are of this type. Such transformations are mainly changes of coordinates. We use this feature of ORION to express all the data related to a location in the coordinates system of the player.

Finally, as for *Pong*, we wish to use algorithms that accept only discrete data as inputs. Therefore we perform a discretization of data using the KMeans algorithm.

4.2.3 CHAMELEON Model

The CHAMELEON model, proposed by Tencé (2011), aims at designing a behavior model for the control of believable characters in video games. This work was conducted in the IHSEV team of Lab-STICC. We therefore had access to the implementation of this model. They define a believable agent as a computer program able to control a NPC in a virtual environment so that other human users in the environment think the avatar is controlled by a human user. They proposed to learn by imitation the layout of environments

with a GNG model. The behavior model is inspired by a model proposed by Le Hy (2007) but they made different choices in order to improve believability. They add to the model the ability to learn by imitation with an Expectation-Maximization algorithm (EM). The proposition makes the model able to learn how to act in the environment rapidly. Stimulus-action associations are made which make the agent look-like a human player. However, this model has some limitations like low action accuracy and lack of planning mechanisms. One of the aims of this thesis is to overcome these problems.

4.2.3.1 Environment Learning

CHAMELEON proposed to learn the environment with a vector quantization method called GNG. The model has been modified to be able to learn continuously on a player without growing indefinitely but being able to extend itself if the teacher begins to use a new part of the environment. Moreover, this model better deals with time series. This model is called Stable Growing Neural Gas (SGNG) (Tencé et al., 2013). The principle of the SGNG is explained in the figure 4.17. The algorithm performs the following steps:

- an input is submitted to the network (a)
- the two closest neurons in the network are selected (b)
- if there is no edge between those, an edge is added, the edge's age is set to 0 otherwise (c)
- error of the closest neuron is increased (d)
- the closest neuron and its neighbors are moved closer to the input (e)
- the ages of all edges of the closest neuron are increased (f)
- edges reaching the maximum age are deleted (g)
- if the closest neuron's error is too high, a new neuron is added between it and its max error neighbor (h) (i)
- errors of all neurons are decreased (j)

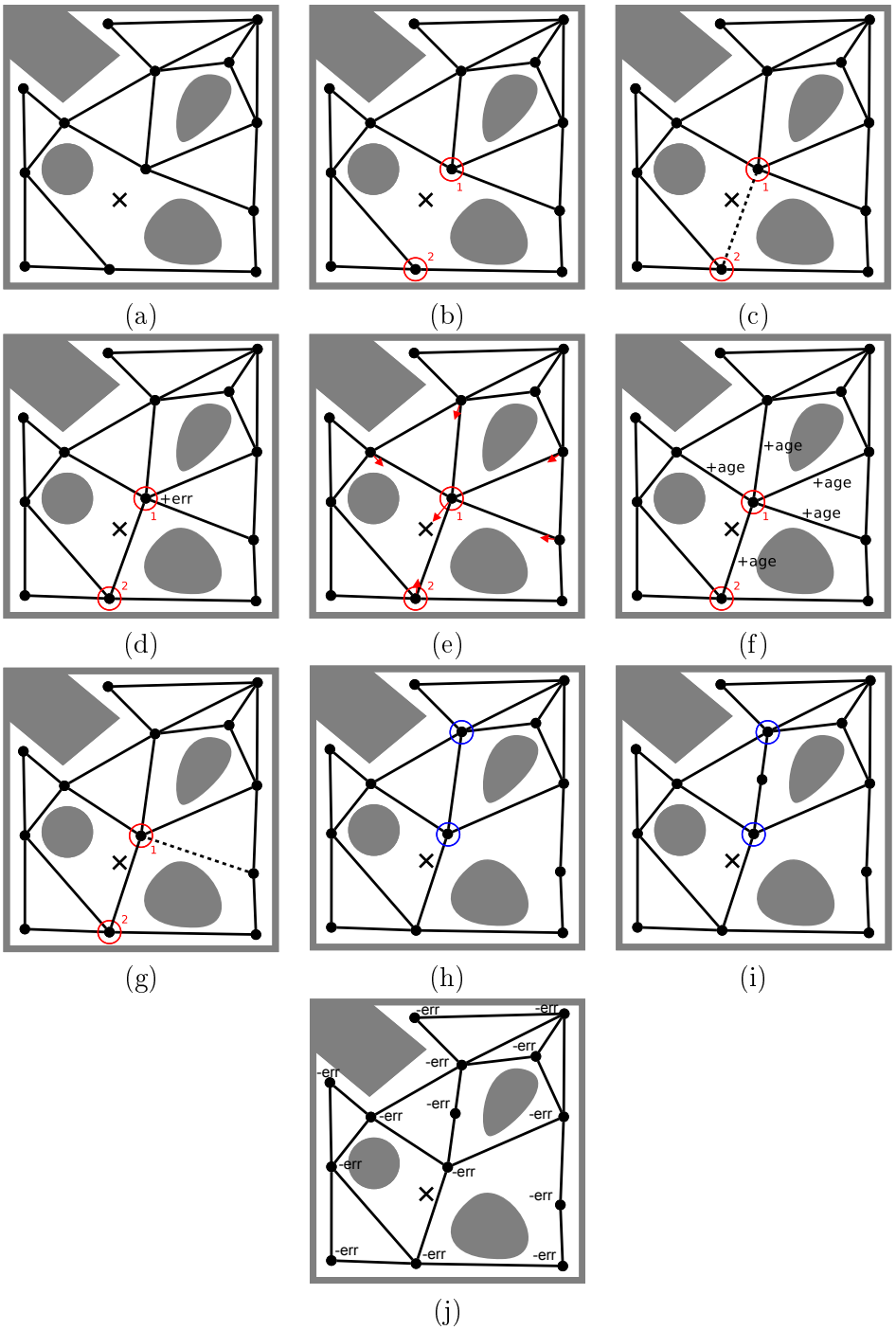


Figure 4.17: *Detailed steps of the SGNG algorithm. The black cross is the input, black circles are the nodes of the SGNG and black lines are the edges of the SGNG. Gray shapes represent the obstacles in the environment.*

4.2.3.2 Behavioral Model

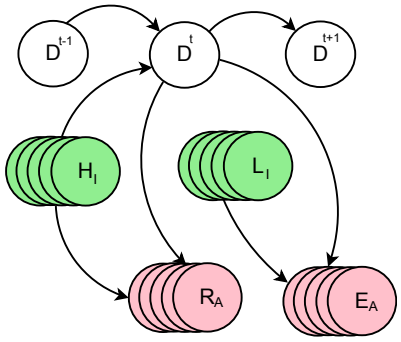


Figure 4.18: CHAMELEON IOHMM model

The CHAMELEON behavioral model is implemented with an Input-Output Hidden Markov Model (IOHMM) (see figure 4.18). The hidden state model corresponds somehow to the strategy of the player. The model inputs are divided into two categories. The first one consists of high level inputs: they are the stimuli necessary to choose strategies (hidden state) and actions. These stimuli represent the internal state of the agent's remaining life points, current weapons, etc. On the other hand, the low level stimuli only influences the choice of actions. These stimuli are typically information about agent's surroundings like positions of the points of interest (enemies, items, etc.). This division allows a semantic refinement of the model. For example, it is not necessary to know exactly where the enemy is located to choose to attack him but it is necessary to have this information to perform the actual attack. This semantic refinement reduces the number of dependencies between random variables and thus reduces learning time. The description of the inputs and outputs of the model is given in appendix B.

The inputs of the model include points of interest characterized primarily by their position. These are low level stimuli, including, for example, navigation points, items and visible enemies. An attention mechanism selects a point of interest which is used to select the actions to perform. This attention mechanism in the model corresponds to a random variable indicating the index of the point of interest, taken into account among all the points of interest in the provided input. Unfortunately, learning the parameters of these random variables is really difficult and is not really efficient in the CHAMELEON model, so probabilities (parameters) of these variables are indicated manually.

4.2.3.3 Implemeting the Chameleon Model with Orion

In order to implement the Chameleon model in Orion, we use the Bayesian inference engine it provides. Each entry is modeled by a random variable defined over the discrete domain used in its implementation. The hidden state of the network, which represents of the strategy or decision of the agent, is defined over a two-element domain. The dependencies between variables meet the recommendations of Tencé between high and low level stimuli and reflective or external outputs. The declaration of this IOHMM is given by the code presented in figure 4.19 (the syntax is specific to Orion).

The model parameters are learned, as in Chameleon model, by the BW algorithm, also implemented in Orion. Since learning the attention mechanism's parameters is not entirely satisfactory in the Chameleon model, we replace, in our implementation, this attention mechanism with a fairly simple *ad hoc* behavior, choosing the point of interest to be considered (in order of priority: the nearest enemy, the nearest item, a waypoint).

```
model unrealbot {
    var POIYaw in {FAR_LEFT, LEFT, CENTER, RIGHT, FAR_RIGHT};
    var POIPitch in {TOP, CENTER, BOTTOM};
    var POIDistance in {CLOSE, MEDIUM, FAR};
    var POIYawSpeed in {FAR_LEFT, LEFT, NONE, RIGHT, FAR_RIGHT};
    var POIPitchSpeed in {TOP, NONE, BOTTOM};


    var LifeArmor in {LOW, MEDIUM, HIGH};
    var NumberOfEnnemies in {ZERO, ONE, TWO, THREE_MORE};
    var CurrentWeaponAmmo in {NONE, LOW, MEDIUM, HIGH};
    var CurrentWeapon in {SHIELD_GUN, ASSAULT_RIFLE, BIO_RIFLE, ONS_MINE_LAYER,
                          SHOCK_RIFLE, MINIGUN, LINK_GUN, FLAK_CANNON,
                          ONS_GRENADE_LAUNCHER, ROCKET_LAUNCHER,
                          ONS_AVRIL, LIGHTNING_GUN, SNIPER_RIFLE, REDEEMER};

    var PreviousDecision in {D1,D2};
    var Decision in {D1,D2} <- NumberOfEnnemies, LifeArmor, CurrentWeaponAmmo,
                               CurrentWeapon, PreviousDecision
    P=random;

    var Motion in {BACKWARD, STOP, FORWARD} <-  POIYaw, POIPitch, POIDistance,
                                                POIYawSpeed, POIPitchSpeed, Decision
    P=random;
    var Strafe in {LEFT, STOP, RIGHT} <- POIYaw, POIPitch, POIDistance,
                                          POIYawSpeed, POIPitchSpeed, Decision
    P=random;
    var Fire in {STOP, FIRE, ALT_FIRE} <- POIYaw, POIPitch, POIDistance,
                                          POIYawSpeed, POIPitchSpeed, Decision
    P=random;
    var Yaw in {FAR_LEFT, LEFT, CENTER, RIGHT, FAR_RIGHT} <- POIYaw, POIPitch,
                                                             POIDistance, POIYawSpeed,
                                                             POIPitchSpeed, Decision
    P=random;
    var Jump in {NO_JUMP, JUMP, DOUBLE_JUMP} <- POIYaw, POIPitch, POIDistance,
                                                 POIYawSpeed, POIPitchSpeed, Decision
    P=random;
    var ChangeWeapon in {SHIELD_GUN, ASSAULT_RIFLE, BIO_RIFLE, ONS_MINE_LAYER,
                         SHOCK_RIFLE, MINIGUN, LINK_GUN, FLAK_CANNON,
                         ONS_GRENADE_LAUNCHER, ROCKET_LAUNCHER,
                         ONS_AVRIL, LIGHTNING_GUN, SNIPER_RIFLE, REDEEMER}
                                        <- POIYaw, POIPitch, POIDistance,
                                           POIYawSpeed, POIPitchSpeed,
                                           NumberOfEnnemies, LifeArmor,
                                           CurrentWeaponAmmo, CurrentWeapon,
                                           Decision
    P=random;
}
```

Figure 4.19: CHAMELEON IOHMM implementation in ORION

4.2.3.4 Evolving Environment Learning

In order to improve the agent's ability to move in the environment, we propose to enrich the data learned by the SGNG. In the CHAMELEON model, the learning environment algorithm rebuilds the navigation graph automatically. This information allows to know which places are reachable in the environment and to compute a path. However, a navigation graph is not sufficient to reproduce the movement of a real player in this environment. A human player with a minimum of experience in the game has, for example, a natural tendency to strafe (move sideways) when reaching the corner of a hallway. This is easily explained by the fact that doing so, the player maximizes his field of vision and can quickly see if an enemy is present in the corridor. Similarly, the navigation graph created by the designers of the game contains special node types indicating for example that a jump is needed at this stage on the way. This information is not learnt by the model.

We propose to add speed and orientation information to be learned by the SGNG. Therefore, at each tick we submit to the SGNG, besides the player's position, his speed and direction. However, this creates an additional difficulty: one of the steps of the algorithm is to submit a datum to the network and to find the two closest data. When the data is a position in space, using the Euclidean distance to find the closest points is totally justified. But when data additionally contains speed and direction, choosing a distance to obtain the desired result becomes more difficult. The orientation is indicated in our data by a unit vector, therefore the use of a simple Euclidean distance would tend to just ignore it from the positions (an unreal unit represents one centimeter). We propose to perform a rescaling by feeding the SGNG with the following distance:

$$\boldsymbol{Distance}((\vec{p1}, \vec{o1}, \vec{v1}), (\vec{p2}, \vec{o2}, \vec{v2})) = \boldsymbol{d}(\vec{p1}, \vec{p2}) + \boldsymbol{\alpha} \boldsymbol{d}(\vec{o1}, \vec{o2}) + \boldsymbol{\beta} \boldsymbol{d}(\vec{v1}, \vec{v2}) \tag{4.1}$$

where $\boldsymbol{d}$ is the regular Euclidean distance and $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are scaling factors.

In order to choose the scale parameter related to the speed, we start from the observation that a player rarely moves backwards when simply navigating (but does, when fighting). The direction of the velocity vector is strongly correlated with the orientation. The norm of the velocity vector is also relatively constant when the player is moving (the character control being done by keyboard, which only contains binary switches). So there seems to be no need to take into account the speed when calculating the distance and we therefore choose to set $\boldsymbol{\beta}$ to zero. On the contrary, the orientation is essential. We expect the vector quantization performed by the SGNG to be able to contain two data having the same position but opposite orientations for example. We choose the parameter $\boldsymbol{\alpha}$ so that criterion is met.

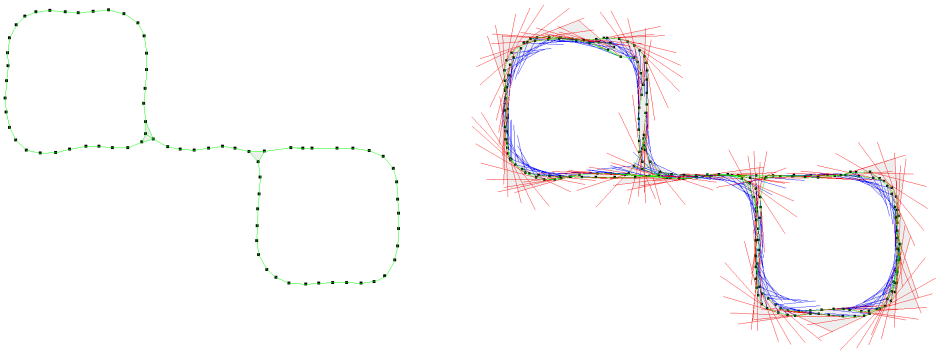


Figure 4.20: SGNG Results on UT3 (Left: CHAMELEON, Right: ORION)

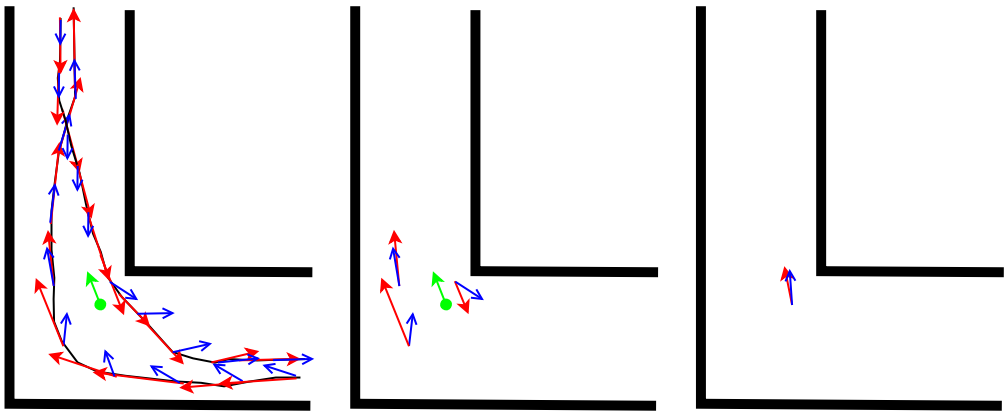
Figure 4.20 illustrates the result. On the left, the SGNG has learned the navigation graph (CHAMELEON version). On the right, the velocity vectors are shown in red and the direction vectors are shown in blue ($\boldsymbol{\alpha} = \mathbf{250}$). We distinguish the difference between the orientation and velocity vectors in the turns, illustrating the strafing performed by the player.

4.2.4 Data Prediction

In section 4.2.1.2, we presented a data collection of low level behavior samples through scenarios previously performed by human players. The problem now is to replicate these behaviors individually.

4.2.4.1 Movement

Section 4.2.3.4 showed how we conducted a vector quantization of the environment, using velocities and orientations in addition to the positions. To reproduce a more realistic navigation behavior, we will use this information with the algorithm KNN. Figure 4.21 illustrates the principle. In this figure, a part of the network is displayed in (a). The point and the green arrow correspond respectively to the current position of the agent and the direction it aims in. As for the construction of the network with the SGNG, the use of KNN requires the use of a distance. Once again we need to perform data scaling. We apply the same distance 4.1 in order to perform this scaling. However, the parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ must be selected differently: knowing the position and the desired direction of the agent, we try to get the speed and direction that the agent should adopt. So this time, we must use the velocity vector (indicating the desired direction) and not the orientation, to compute this distance. Therefore, we choose $\boldsymbol{\alpha} = \mathbf{0}$. In order to set the parameter $\boldsymbol{\beta}$, we apply the following requirement: two data should be far away if their velocities are in opposite directions, even if their position is really close. In figure 4.21, the



(a) (b) (c)

Figure 4.21: Movement regression principle in UT3

image (b) illustrates a possible choice of three nearest data ($\boldsymbol{k = 3}$). The image (c) shows the regression then performed by weighting the closest data by the inverse of the distance to the data submitted to KNN.

4.2.4.2 Long Range Aiming

Aiming is one of the most important elements of FPS games. There are many techniques dedicated for that, and experienced players use advanced ones. The performance of the players in this activity largely determines their overall performance in the game. We chose to make it easier to reproduce this behavior, by learning it from scripted game play in which the player does not move and faces a single enemy. But an experienced player uses, at least partially, the movement for aiming. It is a very commonly seen technique to strafe from side to side without moving the mouse to refine the target. So we will not be able to reproduce the behavior of an experienced player. However, beginners and average players mainly perform a static aiming. So, we focus on reproducing the aiming behavior of such players.

In order to better choose a method to reproduce this behavior, we conduct an exploratory analysis. The inputs and outputs involved in performing of this behavior are rather obvious. The outputs are necessarily rotational speeds (in yaw and pitch) and firing. As inputs, the position of the enemy is essential and it is likely that its speed is necessary too. By performing an exploratory analysis, we found no clear correlation between the enemy orientation and outputs. So we do not use this information as input to learn this behavior. But we found the current weapon is very important as an input. Some weapons are used by continuously pressing the fire button and others by discrete firing. In addition, the velocity of projectiles is different from weapon to weapon.

Therefore, the influence of enemy speed is not the same from one weapon to another. We identify, regardless of the weapon used, the following correlations between:

- the Y position of the enemy in the player's coordinate system (right or left) and the yaw rotation speed
- the Z position of the enemy in the player's coordinate system (top or bottom) and the pitch rotation speed
- the Y enemy speed in the player's coordinate system (right or left) and the yaw rotation speed

The differences in behavior according to the weapon suggest to process different training sessions for each of them. We learn these behaviors with regression algorithms. We conducted this learning through FFNN, KNN and SVM algorithms. The quantitative results appeared to be quite similar from one algorithm to another but the resulting behaviors were not really correlated to the MSE.

To compare the artificial behaviors to those of the human player, we reproduce them in our test case scenario. When collecting the data for the weapon called LinkGun, the human player scored 50 points and was killed 10 times (so the bot scores 10 points), winning the game. We have reproduced this scenario using our behavior instead of the player's. The best behavior, obtained with a multilayer perceptron, killed the bot 32 times and was eliminated 50 times, losing the game. Our behavior is therefore less efficient than the player.

4.2.4.3 Close Combat

Close combat is quite common in UT3. The arenas are often made of tortuous corridors, where one often encounters enemies at turning points. Some weapons are more appropriate for this kind of confrontation. Reflexes are necessary but an unpredictable behavior is the key to make the task of the enemy harder.

By analyzing the data from our scenario in which the player is in contact with an enemy in a small room, it was difficult to find relevant information in the context of the reproduction of this behavior. Changes in direction and jumps are common, without being obviously correlated with the slightest stimuli. The player, however, constantly tries to face the enemy. Strafing and backward runs are often used rather than the forward movement.

We tried to reproduce this behavior using standard regression algorithms that we have used for the aiming behavior. However, we have not been able to obtain convincing results with this method. This is not surprising because the exploratory analysis shows that there was no obvious dependencies between inputs and outputs. These algorithms are used to approximate a mathematical function that, for a given input always provides the same output. However, in our data set, this is clearly not the case.

Therefore, we used a IOHMM to reproduce this behavior. A probabilistic model is indeed more suited to this type of data because it provides a certain unpredictability. We used a network very similar to the CHAMELEON model, applying the recommendations on variable dependencies. The behavior resulting from this learning seems quite satisfactory.

4.2.5 Behavioral Model

We have shown how we reproduce three low level actions: shoot the enemy from afar, fighting hand-to-hand and navigating in the environment. Now we have to offer a complete behavior, using these actions in the behavioral model of ORION. The overall behavior is implemented with a BT using common methods from the video game industry.

Figure 4.22 shows a simplified version of the BT implementing the overall behavior. In this figure, the green boxes represent reference decorator nodes. They are used to name a sub-tree in order to reference that sub-tree elsewhere in the BT. They are only present here for the sake of clarity. The behavior starts to transform the input data (discretization and change of basis). The implementation details of this node are not shown in the figure. Then, the selected node is in charge of the choice of the low level behavior to execute. The implementations of CloseCombat and AimAndShoot nodes is quite simple, since they test if an enemy is present (and close enough in the case of CloseCombat), select the nearest enemy (by adding a variable in the execution context) and perform the regression with the algorithm having been previously trained. Both behaviors then delete variables used by the navigation behavior from the execution context. If no enemy is present, the navigation behavior is executed. It consists in choosing a destination if needed (and puting it in the context of execution), calculating the path to this destination if needed too, selecting the navigation point to which to go and performing regression movement with KNN. The BT is scanned depth-first on each tick.

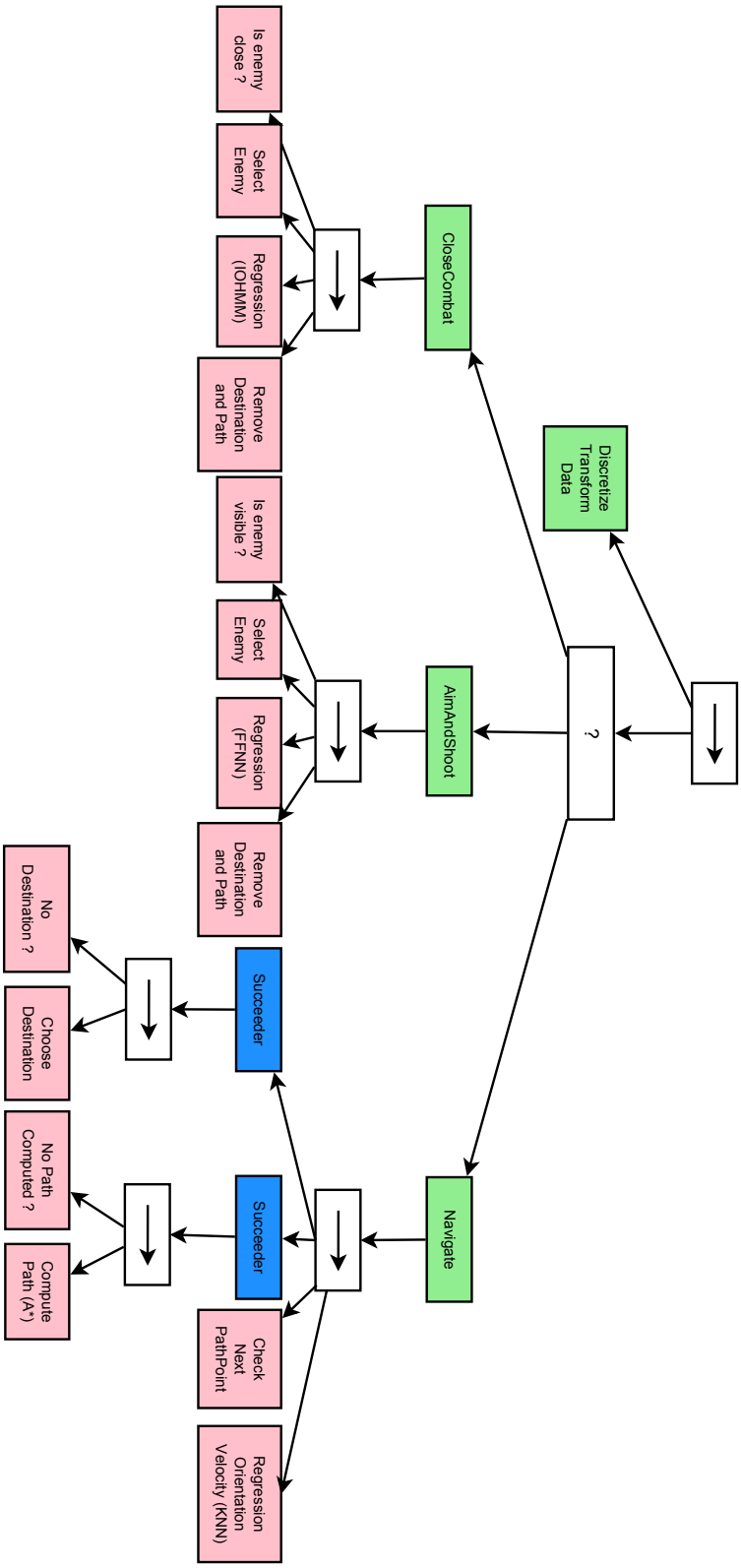


Figure 4.22: ORION BT of our UT3 Agent

4.3 Conclusion

In this chapter, we illustrated the use of our model to produce Artificial Intelligence in *Pong* and Unreal Tournament 3 games using machine learning techniques. We have shown how the exploratory data analysis can help to identify noticeable patterns like invariant between players and also provides help to make the choice of learning techniques to use.

We also showed how some of the supervised and unsupervised learning algorithms can be used as part of the creation of an AI in video games. Unsupervised learning, for example, helped us to automatically reconstruct the environment and the information on how the players move in it. Supervised learning has allowed us to reproduce relatively simple actions that can be used as primitives to implement more elaborate behaviors.

While the behaviors conducted with our model are neither more efficient nor more believable than behavior implemented with far more widely used methods in this industry, their implementation shows that learning methods from the world of research can help the design of AI in video games, easing the designer's workload and providing it with relevant information on human player behavior.

Contents of Chapter 5

Chapter 5

Conclusion

Big things have small beginnings.

David - Prometheus (2012)

Summary

In this chapter, we first summarize this thesis. We then discuss the proposals we have made, their contributions and their weaknesses. We provide suggestions for future works to overcome them.

Introduction

In this thesis, we proposed a general tool for data mining and a behavioral model design to build video game characters based on data from real players. In this chapter we examine the strengths and weaknesses of our proposals and suggest some areas for improvement.

Plan

In this chapter, we start by summarizing this thesis (section 5.1). We discuss the proposals put forward in this thesis (section 5.2), their contributions to the problem of designing Artificial Intelligence in video games (section 5.2.1) and the main weaknesses of our proposals (section 5.2.2). We then offer some suggestions for overcoming them (section 5.3).

5.1 Summary of the Thesis

We started this thesis by identifying the current and future needs of Artificial Intelligence (AI) in video games by means of a historical review of video games. We then discussed the purpose of AI. How is Artificial Intelligence used? What features are used to implement it? What additional constraints are necessary?

With this review we were able to show how imitating real players seemed to be a promising and fruitful strategy.

We then studied the development of a generic software solution for AI design in video games. We first listed the requirements such a system would need. The first requirements are classic constraints in video game development. Next, we focused on the characteristics of human behavior in a video game context. These features helped us to define requirements regarding the expressiveness of the models. Since we want to learn these behaviors, we are also interested in the requirements of the learning process itself. These different requirements allowed us to put the relevance of different models into perspective in our context. We want to imitate realistic human behavior, therefore we need to operate with data from real humans operating in the game environment. The field of Data Mining offers an interesting range of techniques that can be useful. Weber and Mateas (2009) have already successfully used some data mining techniques to implement a real time strategy game bot. Thus we evaluated various data mining techniques, according to the requirements defined previously. We concluded that all of these techniques may be relevant and that our solution must allow the use of several of them. However, these techniques are far from being sufficient to fulfill all our requirements.

Most AI techniques used in video games can be integrated into different kinds of global architectures. We studied the following types of architectures: procedural, rule-based, structured and multi-agents. We concluded from this study that structured architectures are best suited to fulfilling our requirements. We then presented the ORION model. This model is composed of a structural and a behavioral part. The structural part allows datasets to be handled generically and even enables semantics to be added to data. The behavioral part is based on behavior trees. Together, these two sides of the model satisfy many of the needs expressed before. Moreover, we propose an extension that allows to build an AI from data retrieved from real players. When using behavior trees, data are usually stored in a key-value dictionary used by every behavior. The structural model helps to streamline inputs and outputs used by the model. ORION offers typed data, it is therefore possible to maintain the semantics of the data throughout the design as well as to check the consistency of the data used by each behavior. Thanks to our implementation of the model, we demonstrated the flexibility of our approach by manipulating and visualizing various data sets from a database known as UCI[1]. The behavioral model, although based on behavior trees, can be used to handle behaviors performed by machine learning both online (during the execution of the game) and offline (during the design phase). These behaviors can produce additional knowledge from data (such as automatic learning of

[1] http://archive.ics.uci.edu/ml/datasets.html

the environment) or automatically partition the set of sequences in order to identify different strategies used by the player.

Later, using ORION, we implemented two applications. We applied data mining techniques to the video game *Pong*. We analyze the data from games played by human players. We showed that exploratory analysis allows us to identify various strategies implemented by players. We also identified the evolution of these strategies during gameplay along with invariants between players. We continued using unsupervised methods of analysis to segment our data into homogeneous groups. We tried to identify the strategy of the player corresponding to a given situation. We saw that we are, to some extent, able to perform this clustering automatically. However, we evoked reservations about the applicability of this method. Then we used supervised learning techniques to reproduce part of the behavior (low level actions). To carry out this task, we first annotated game sequences with a semi-automatic method using rules. Each identically annotated sequence was then used as training data for the regression algorithms tested. We evaluated the performance of each algorithm. We used a classification algorithm in order to choose the most appropriate low level action to execute, thus producing the overall behavior. Subsequently, we used ORION to implement a bot for the game Unreal Tournament 3. We started by implementing the CHAMELEON model (Tencé, 2011) with ORION. Then, we proposed an improvement of the CHAMELEON method to learn the environment automatically. Still using our model, we used that improvement to make the bot movements more in line with those of a human player. We also learned aiming and close combat behaviors through supervised learning techniques. Finally, we implemented the overall behavior using the ORION BT, integrating *ad hoc* features in order to manage aspects of behavior that have not be learned automatically.

5.2 Discussion

In this section, we discuss the contributions and weaknesses of our proposal.

5.2.1 Contributions

5.2.1.1 Data Mining Tool

This thesis was conducted in order to offer a flexible solution to designing AI in video games based on data from real players. Data mining seemed to us to be a field of research which could provide many useful techniques in this context. We therefore had to implement and test a large number of these techniques and to provide a general tool for Data Mining, similar to software like WEKA, Orange Data Mining or ELKI. While the primary reason for these implementations (rather than using one of these programs) was to obtain

deeper insight into these techniques, it soon became clear that these tools do not allow for the addition of semantics to data. Datasets are usually limited to a set of discrete or continuous attributes. However, it seemed important to rely on data semantics. First, because using semantics makes it possible to visualize this data in a more understandable way. Furthermore, some data transformations are only applicable to data with specific semantics. Existing tools do not offer efficient means of data visualization. This therefore led us to develop a model with advanced visualization abilities. By using our tool, it is now possible to represent a wider variety of data. Finally, this tool offers an abstraction of tasks that are achievable through Data Mining techniques.

5.2.1.2 Behavior Tree with Data Mining Possibilities

The behavioral model of ORION aims to complement the CHAMELEON model (Tencé, 2011). One of the main flaws of CHAMELEON was the poor accuracy of low-level actions. This model used only discrete data, so it was not possible to accurately model behaviors such as that of targeting the enemy in UT3. This is because perceptions and actions were discrete and therefore the expressiveness was limited to actions like slightly turning to the left or right, going forward or backward, etc. Under these conditions, the resulting behavior was not able to accurately track the target. By using Data Mining, our model makes it possible to learn these behaviors much more reliably.

Our behavioral model is based on the Behavior Trees. This way, the behavior designer has complete freedom to achieve goal-oriented behavior. The integration of standard problem-solving algorithms such as path finding is available in our model. Low-level actions such as targeting behavior or evasion maneuvers can be learned instead of being hard-coded, thus significantly reducing the workload and saving time. Furthermore, our model proposes an online learning mechanism.

5.2.1.3 Environment Learning Improvement

In the CHAMELEON model, the environment learning mechanism consisted of vector quantization using the SGNG algorithm. Using this algorithm it was possible to learn the environment's navigation graph online. This navigation graph allows the agent to find its way through the environment, but does not provide any information about how to actually perform the movement. For example, UT3 players arriving at the corner of a hallway generally use a lateral displacement (strafing) to see the end of the corridor and detect potential enemies as quickly as possible. Another example is jumping over a ravine. The navigation graph does not give any information about the need to jump in order to move from one platform to another. So we added velocity and orientation information to the learning process through the SGNG algorithm.

This information can help fill these gaps. Using all these data (positions, orientations and velocities) with the KNN algorithm, it was possible to obtain movements that reproduce the strafing behavior of the human players at the turn points of the corridors.

5.2.2 Weaknesses

The proposals in this thesis are still far from providing a complete solution for data mining and learning by imitation and minimizing the workload needed for its implementation effectively. In this section, we present the main shortcomings of our work.

5.2.2.1 Data Mining

While ORION tries to structure the Data Mining techniques by providing a level of abstraction of these methods, we believe it would be better if this structure were more in-depth. As we have seen, Data Mining is a very broad and multidisciplinary field. While our model is not sufficiently fit for all the problems encountered in Data Mining, we identify a series of possible improvements to ORION. In ORION, we do not model the parameters of algorithms; in fact, we use Java introspection mechanisms to offer one of these settings to the user by using the GUI. Yet, such modeling would facilitate the automatic search for optimal parameters and provide non-expert users with more information in order to better select these parameters.

ORION does not organize the flow of data processing as is possible in Orange Data Mining. This tool indeed offers a user-friendly interface for defining the processing sequence to be done. This kind of approach is commonly included in many tools such as in the 3D modeling software, Blender [2]. This is an important shortfall in ORION.

5.2.2.2 Behavioral Model

We chose the BT model as the basis of our behavioral model. This is a deliberate choice insofar as this model is both very flexible and that it enables a representation of the behavior that is easier to handle. But the BTs are a programming paradigm. Therefore, our model can currently only be used by experts with programming skills. Developing a behavior still requires a significant effort, although the proposed extension of the model does, to some extent, decrease it. We hope that our proposals offer a progression toward the final goal which is automatic and complete learning of a behavior from the traces of a human. However, our model is still far from performing it.

[2] http://www.blender.org/

To achieve this goal, the low-level actions in the traces must be identified automatically. In our applications, this task was performed by an expert or using learning scenarios containing only traces of these low-level actions. However, Data Mining techniques can be used to perform this task. We have obtained encouraging results in splitting sequences of gameplay in *Pong* but we were not able to provide usable results.

Finally, while the use of human player traces is an essential source of data to reach our goal, we have not considered other data sources. It seems reasonable to assume that a particular player attempts to win when playing a game. However, our model does not take performance into account. Behavior resulting from learning should be as good as the "teacher" player and, consequently, the performance of the resulting behavior should be taken into account in learning.

5.2.2.3 Evaluation

The applications on which we used our model are relatively simple. *Pong* has been very useful for testing our model on a case that does not require major expressiveness. Our application on UT3 was mostly to show the integration of CHAMELEON within our architecture. This application has also allowed us to demonstrate that it was possible to enrich this model with more traditional AI methods and the use of learning methods to achieve low-level behaviors. It would nonetheless be interesting to use ORION for a more ambitious application. In addition, the behaviors we made have not been evaluated. A study of the believability of obtained behaviors and the development time saved by using our model, are still necessary.

5.3 Future Work

The limits of ORION, identified in section 5.2.2, led us to suggest ways in which our model could be improved. The proposed improvements are identified in green in figure 5.1 within the ORION workflow.

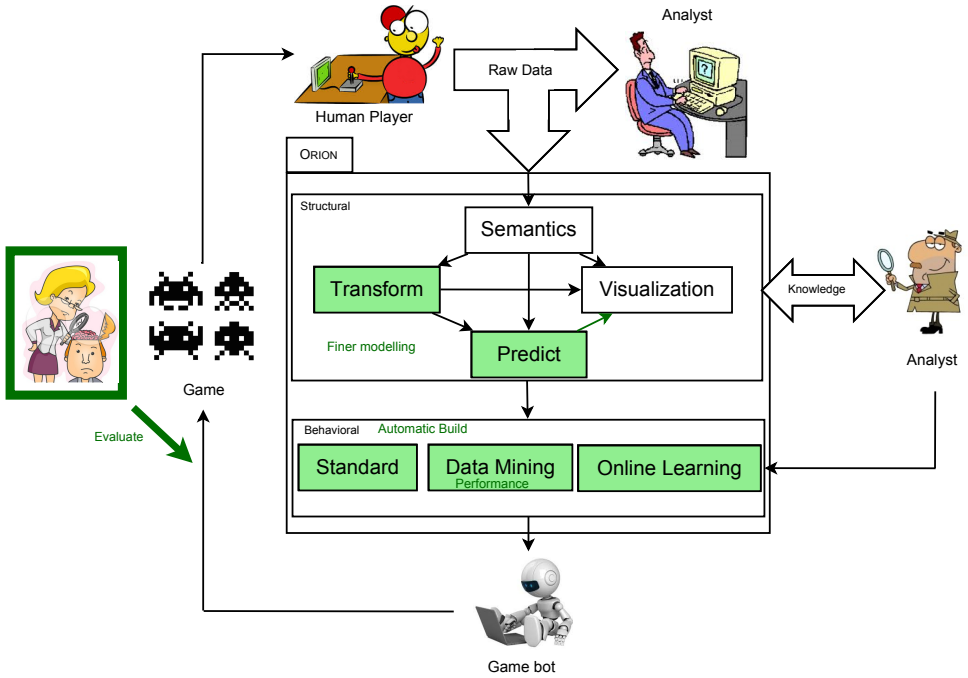


Figure 5.1: Future Work (green) represented in the ORION Workflow

5.3.1 Data Mining

Our implementation of the structural model of ORION could be greatly improved with finer modeling of Data Mining techniques. Modeling algorithm parameters would provide additional information, in order to automatically search for optimal values of these parameters during a classification or regression process. Such research may be performed by optimization techniques such as simulated annealing or a genetic algorithm for example. By observing the existing tools such as WEKA, it appears to us that the use of Data Mining tools is quite complex when users are unfamiliar with the algorithms they try to use. Better modeling of algorithm parameters would therefore provide more relevant information to the user about their meaning, generally acceptable values, their influence on the result, and provide a more educational tool.

Also with a view to improving the educational qualities of ORION, it would be interesting to propose a step by step execution of the iterative algorithms

implemented. This would provide, on the one hand, a better understanding of the algorithm process to non-specialist users, and on the other hand, it would facilitate the choice of some parameters. Moreover, ORION does not currently provide prediction visualization abilities. It would be an interesting feature to be able to graphically specify the input and to see the prediction results. As an example, using the movement regression we proposed in UT3, the user would graphically specify the position and the direction of the bot and be able to visualize the resulting velocity and orientation.

5.3.2 Behavior Model

Many issues still need to be resolved before our behavioral model can automatically and completely learn both credible and effective behaviors without human intervention. The first opportunity for improvement is probably automatic identification of low-level behavior in the traces of the player. Temporal clustering is a difficult task but work done as part of the video segmentation, such as that of (Nguyen, 2012), seems to provide good results. While we have not been able to effectively adapt these techniques to our situations, it nevertheless seems to us that some results are encouraging and that it is worth pursuing.

The addition of a reinforcement learning mechanism to our model seems to be a possible avenue for improvement. Taking into account the performance of the behavior seems a very useful source of information in order to obtain more convincing behavior. However, the choices made during this thesis do not directly support such an integration. It is therefore possible that some proposals in this thesis should be challenged in order to accommodate this change.

Finally, an automatic construction of BTs could be seen as one of the last steps required for creating a behavior by using traces and without human intervention. But this task still seems difficult to access and therefore seems to be a distant prospect of improvement.

5.3.3 Evaluation

Finally, this research aims to address the problem of creating believable virtual characters (Buche, 2012) in video games. Believability is very complex and subjective. According to Thomas and Johnston, two major Disney animators, the objective of believable characters is to give the illusion of life (Johnston and Thomas, 1981). Reidl's definition is more accurate: "Character believability refers to numerous elements that allow a character to achieve the "illusion of life", including but not limited to personality, emotion, intentionality, and physiology and physiological movement" (Riedl and Young, 2005, p.2).

Loyall believes that such a character "provides a convincing portrayal of the personality they [the spectators] expect or come to expect" (Loyall, 1997). If we want to apply the definition of believability to video games, things become even more complex. Players can be incarnated into avatars and can interact.

The issue of believability is to procure the illusion that the virtual players are controlled by a human player (Livingstone, 2006). As believability is subjective, evaluation is a crucial and complex step. The Turing test is still considered as a reference benchmark to evaluate believability (Turing, 1950). In its standard interpretation, a judge should discuss with either a human or a machine using only text. If after a certain period of time, the judge cannot tell if the entity is artificial or not, the machine is said to have passed the test.

The objective of this test was to evaluate intelligence, but it was much criticized (Searle, 1980; Hayes and Ford, 1995). This criticism, however, is not about believability. Many parameters in believability evaluation methods have to be set (Mac Namee, 2004; Livingstone, 2006), for example the number of questions/answers. The Turing test has only one question and one yes/no answer while others have many questions and graduated responses. The original choice may be too restrictive and the others may lead to uncertain responses. Another problem is the calculation of the overall score of believability because, in case of multiple questions, experimenters can influence the results. To add more objectivity, it is possible to have a relative notation of the score of the participant ("Is participant A more believable than participant B?"). It is also necessary to decide whether the judges participate in the gameplay or are just spectators. While players can actively test evaluated characters, viewers are more focused on the entities to evaluate and may notice more details in behavior. Finally, the choice of judges is particularly important. Cultural background (Mac Namee, 2004) and experience level (Bossard et al., 2009) may have a significant impact on believability scores. Such a complex study therefore needs to be carried out by psychologists, since such evaluation involves humans and their feelings about the observed behavior.

5.4 Publications

This section presents the articles published as part of this thesis.

- Tencé, F., Gaubert, L., **Soler, J.**, De Loor, P., & Buche, C. (2013). Stable growing eural gas: A topology learning algorithm based on player tracking in video games. Applied Soft Computing, 13(10), 4174–4184.

- Tencé, F., Gaubert, L., **Soler, J.**, De Loor, P., & Buche, C. (2013). CHAMELEON: Online learning for believable behaviors based on humans imitation in computer games. Computer Animation and Virtual Worlds, 24(5), 477–496.

- **Soler J.**, Tencé F., Gaubert L., & Buche C. (2013). Data Clustering and Similarity. In Proceedings of the Twenty-Sixth International Florida Artificial Intelligence Research Society Conference (pp. 492 – 495).

Bibliography

Achtert, E., Kriegel, H.-P., and Zimek, A. (2008). ELKI: a software system for evaluation of subspace clustering algorithms. In *Scientific and Statistical Database Management*, pages 580–585. 24, 48

Aggarwal, C. C., Hinneburg, A., and Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional space. *Database Theory - ICDT 2001*, pages 420–434. A3

Agrawal, R., Imielinski, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data - SIGMOD '93*, number May, pages 207–216, New York, New York, USA. ACM Press. 23

Agrawal, R., Lin, K.-I., Sawhney, H. S., and Shim, K. (1995). Fast similarity search in the presence of noise, scaling, and translation in time-series databases. *IBM Almaden Research Center*. A10

Ankerst, M., Breunig, M. M., Kriegel, H.-P., and Sander, J. (1999). OPTICS: ordering points to identify the clustering structure. *ACM SIGMOD Record*. 37, A10

Arthur, D. and Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. 35

Baum, L. E., Petrie, T., Soules, G., and Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The annals of mathematical statistics*, 41(1):164–171. 33

Bengio, Y. and Frasconi, P. (1995). An input output HMM architecture. *Advances in neural information processing systems*, pages 427–434. 32

Berndt, D. J. and Clifford, J. (1994). Using dynamic time warping to find patterns in time series. *KDD workshop*, pages 359–370. A9

Bollobás, B., Das, G., Gunopulos, D., and Mannila, H. (1997). Time-series similarity problems and well-separated geometric sets. *Proceedings of the*

thirteenth annual symposium on Computational geometry - SCG '97, pages 454–456. A10

Booth, M. (2009). The ai systems of left 4 dead. In *Keynote, Fifth Artificial Intelligence and Interactive Digital Entertainment Conference*. 10

Bossard, C., Kermarrec, G., and De Loor, P. (2009). Sport, réalité virtuelle et conception de simulations participatives. Illustration dans le domaine du football avec le simulateur CoPeFoot. *Intellectica*. 109

Buche, C. (2012). *Adaptive behaviors for virtual entities in participatory virtual environments*. Habilitation à diriger des recherches, Université de Bretagne Occidentale - Brest. 108

Canales, F. and Chacón, M. (2007). Modification of the growing neural gas algorithm for cluster analysis. *Progress in Pattern Recognition, Image Analysis and Applications*, 4756(3659):684–693. 36

Chang, C.-c. and Lin, C.-j. (2011). LIBSVM : A Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2:1–39. 59

Charalampidis, D. and Muldrey, B. (2009). Clustering using multilayer perceptrons. *Nonlinear Analysis: Theory, Methods & Applications*, 71(12):e2807–e2813. 35

Chen, M.-S., Han, J., and Yu, P. S. (1996). Data mining: an overview from a database perspective. *Knowledge and Data Engineering, IEEE Transactions on*, 8(6):866–883. 23, A1

Coifman, R. R. and Lafon, S. (2006). Diffusion maps. *Applied and Computational Harmonic Analysis*, 21(1):5–30. 26

Csikszentmihalyi, M. (1991). *Flow: The psychology of optimal experience*. 5

Das, G., Gunopulos, D., and Mannila, H. (1997). Finding similar time series. *Principles of Data Mining and Knowledge Discovery*, 1263:88–100. A10

Defays, D. (1977). An efficient algorithm for a complete link method. *The Computer Journal*, 20(4):364–366. 36, A7

Demšar, J., Curk, T., Erjavec, A., Gorup, v., Hočevar, T., Milutinovič, M., Možina, M., Polajnar, M., Toplak, M., Starič, A., Štajdohar, M., Umek, L., Žagar, L., Žbontar, J., Žitnik, M., and Zupan, B. (2013). Orange: data mining toolbox in python. *The Journal of Machine Learning Research*, 14(1):2349–2353. 24

Dhillon, I. S., Guan, Y., and Kulis, B. (2004). Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 551–556. 35

Doherty, D. and O'Riordan, C. (2006). The Design Goals and Implementation of AI in Modern Computer Games. Technical report. 17

Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data mining*, pages 226–231. 37, A10

Evans, R. (2001). The Use of AI Techniques in Black & White. 8

Fayyad, U., Piatetsky-shapiro, G., and Smyth, P. (1996). From Data Mining to Knowledge Discovery in Databases. pages 37–54. 21, 22

Freund, Y. and Schapire, R. E. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139. 33, 34

Friedman, N. (1998). The Bayesian structural EM algorithm. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 129–138. 32

Fritzke, B. (1994). Growing cell structures - A self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460. 36

Fritzke, B. (1995a). A growing neural gas network learns topologies. *Advances in neural information processing systems*, 7:625–632. 36, A10

Fritzke, B. (1995b). Growing grid - A self-organizing network with constant neighborhood range and adaptation strength. *Neural Processing Letters*, 2(5):9–13. 36

Goldsmith, T. T. J. (1948). Cathode-ray tube amusement device. 3

Grand, S. and Cliff, D. (1998). Creatures: Entertainment software agents with artificial life. *Autonomous Agents and Multi-Agent Systems*, 57:39–57. 7

Guénoche, A., Hansen, P., and Jaumard, B. (1991). Efficient algorithms for divisive hierarchical clustering with the diameter criterion. *Journal of classification*, 8:5–30. 36

Hall, M., National, H., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA Data Mining Software : An Update. *SIGKDD Explorations*, 11(1):10–18. 48

Hand, D. J. (2007). *Principles of data mining.*, volume 30. 21

Hart, P., Nilsson, N., and Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2). 7

Hayes, P. and Ford, K. (1995). Turing test considered harmful. *International Joint Conference on Artificial Intelligence*, pages 972–977. 109

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558. 29

Isla, D. (2005). Handling complexity in the Halo 2 AI. *Game Developers Conference*. 9, 18

Jain, A. K. (2010). Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 31(8):651–666. 35

Johnston, O. and Thomas, F. (1981). *Disney Animation: The Illusion of Life*. 108

Karypis, G., Han, E.-H., and Kumar, V. (1999). Chameleon: hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75. 36, A8

Kaufman, L. and Rousseeuw, P. J. (1987). Clustering by means of medoids. *Statistical data analysis based on the L1-norm and related methods*, pages 405–416. 35

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59–69. 36, A10

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480. 36

Koskela, T., Lehtokangas, M., Saarinen, J., and Kaski, K. (1996). Time Series Prediction with Multilayer Perceptron , FIR and Elman Neural Networks. In *In Proceedings of the World Congress on Neural Networks*, pages 491–496. 29

Laird, J. E. and Duchi, J. C. (2000). Creating human-like synthetic characters with multiple skill levels: A case study using the soar quakebot. *Simulating Human Agents, Papers from the 2000 AAAI Fall Symposium*, pages 75–79. 18, 46

Le Hy, R. (2007). *Programmation et apprentissage bayésien de comportements pour des personnages synthétiques – application aux personnages de jeux vidéos*. PhD thesis, Institut National Polytechnique de Grenoble. 88

Livingstone, D. (2006). Turing's test and believable AI in games. *Computers in Entertainment*, 4(1):6. 17, 18, 19, 109

Lloyd, S. P. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137. 35

Loyall, A. B. (1997). *Believable Agents : Building Interactive Personalities Thesis Committee :*. PhD thesis. 19, 109

Mac Namee, B. (2004). Proactive Persistent Agents: Using Situational Intelligence to Create Support Characters in Character-Centric Computer Games. 109

Macqueen, J. (1966). Some methods for classification and analysis. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 233, pages 281–297. 35

Martinetz, T. M., Berkovich, S. G., and Schulten, K. J. (1993). 'Neural-gas' network for vector quantization and its application to time-series prediction. *Neural Networks, IEEE Transactions on*, 4(4):558–569. 36

McLaren, K. (2008). The Development of the CIE 1976 (L* a* b*) Uniform Colour Space and Colour-difference Formula. *Journal of the Society of Dyers and Colourists*, 92(9):338–341. 22

Mercer, J. (1909). Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London*, pages 69–70. 30, A4

Nguyen, M. H. (2012). *Segment-based SVMs for Time Series Analysis*. PhD thesis, Carnegie Mellon University. 108

Orkin, J. (2006). Three states and a plan: the AI of FEAR. *Game Developers Conference*, pages 1–18. 9

Querrec, R., Vallejo, P., and Buche, C. (2013). MASCARET: creating virtual learning environments from system modelling. In *Proc. SPIE 8649, The Engineering Reality of Virtual Reality 2013*, volume 8649, pages 864904–864911. SPIE. 46

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, pages 81–106. 8, 77

R Developement Core Team (2008). *R: A Language and Environment for Statistical Computing*, volume 1. R Foundation for Statistical Computing, Vienna, Austria. 24

Riedl, M. O. and Young, R. M. (2005). An objective character believability evaluation procedure for multi-agent story generation systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3661 LNAI:278–291. 108

Rivière, J., Pesty, S., and Adam, C. (2012). Un ACA sincère comme compagnon artificiel. In *Workshop on Affective Compagnon Artificiel, Interaction*. 47

Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science (New York, N.Y.)*, 290(5500):2323–6. 26

Russel, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd editio edition. 33

Salvador, S. and Chan, P. (2007). FastDTW: Toward Accurate Dynamic Time Warping in Linear Time and Space. *Intelligent Data Analysis*, 11(5):561–580. A9

Sammon, J. W. (1969). A Nonlinear Mapping for Data Structure Analysis. *IEEE Transactions on Computers*, C-18(5):401–409. 27

Schölkopf, B., Smola, A., and Müller, K.-R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319. 25, A4

Searle, J. R. (1980). Minds, brains, and programs. *Behavioral and Brain Sciences*, 3(03):417–424. 109

Shi, L.-K., He, P.-L., Liu, B., Fu, K., and Wu, Q. (2005). A robust generalization of isomap for new data. *2005 International Conference on Machine Learning and Cybernetics*, 3(August):18–21. 27

Sibson, R. (1973). SLINK: an optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34. 36, A7

Smuts, A. (2005). Are Video Games Art? *Contemporary Aesthetics*, 3. 1

Smyth, P. (1997). Clustering sequences with hidden Markov models. *Advances in neural information processing systems*, 9:648–654. 37

Stokes, M., Anderson, M., Chandrasekar, S., and Motta, R. (1996). A Standard Default Color Space for the Internet - sRGB. 22

Swayne, D. F., Buja, A., and Lang, D. T. (2004). Exploratory Visual Analysis of Graphs in GGobi. In *COMPSTAT*, number Dsc, pages 477–488. 24, 48

Tencé, F. (2011). *Probabilistic Behaviour Model and Imitation Learning Algorithm for Believable Characters in Video Games*. Phd, Université de Bretagne Occidentale. iv, 17, 18, 19, 69, 70, 87, 103, 104

Tencé, F., Gaubert, L., Soler, J., De Loor, P., and Buche, C. (2013). Stable growing neural gas: A topology learning algorithm based on player tracking in video games. *Applied Soft Computing*, 13(10):4174–4184. 36, 88

Tenenbaum, J. B., de Silva, V., and Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science (New York, N.Y.)*, 290(5500):2319–23. 26

Torgerson, W. S. (1952). Multidimensional scaling: I. Theory and method. *Psychometrika*, 17(4):401–419. 25

Trinh, T.-H., Chevaillier, P., Barange, M., Soler, J., De Loor, P., and Querrec, R. (2011). Integrating Semantic Directional Relationships into Virtual Environments: A Meta-modelling Approach. In *JVRC11: Joint Virtual Reality Conference of EGVE - EuroVR, Nottingham, UK, 2011.*, pages 67–74. 47

Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind*, 59(236):433–460. 109

Vapnik, V., Golowich, S. E., and Smola, A. (1996). Support vector method for function approximation, regression estimation, and signal processing. In *Advances in Neural Information Processing Systems 9*, pages 281–287. A4

Viola, P. and Jones, M. J. (2004). Robust Real-Time Face Detection. 57(2):137–154. 34

Viterbi, a. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269. 32

Vondrick, C., Khosla, A., Malisiewicz, T., and Torralba, A. (2013). HOGgles: Visualizing Object Detection Features. *2013 IEEE International Conference on Computer Vision*, pages 1–8. 48

Weber, B. G. and Mateas, M. (2009). A data mining approach to strategy prediction. *CIG2009 - 2009 IEEE Symposium on Computational Intelligence and Games*, pages 140–147. 15, 102

Witten, I. H. and Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition*. 24

Zhang, R. and Rudnicky, A. I. (2002). A large scale clustering scheme for kernel K-Means. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 4, pages 289–292. IEEE Comput. Soc. 35

Contents of Chapter A

Appendix A

Data Clustering and Similarity

This appendix was published in the 26th International Florida Artificial Intelligence Research Society Conference

FLAIRS'13

A.1 Introduction

Data clustering is an important part of data mining. This technique is used in many fields such as biological data analysis or image segmentation. The aim is to identify groups of data known as clusters, in which the data are similar.

Effective clustering maximizes intra-cluster similarities and minimizes inter-cluster similarities (Chen et al., 1996). Before defining inter- and intra-cluster similarities, first we must define the similarity between a data pair. There are many different ways of defining these similarities and depending on the chosen method, the results of the cluster analysis may strongly differ.

So how can we reliably choose one definition over another? The aim of this article is to provide clues in order to answer this question by studying different definitions of similarity. We will study the following elements separately: distance between two pieces of data and inter-and intra-cluster distances.

In section A.2, we will discuss the notion of distance between two pieces of data. We shall examine different distances which can be used in the majority of clustering algorithms and their relevance depending on the problem at hand. We shall see in section A.3 how the use of different inter/intra-cluster distances can dote the resulting clusters with certain properties. Finally, we will discuss how our study might be of merit for unresolved issues.

A.2 Data Similarity

Generally speaking, data similarity is evaluated using the notion of distance. In this section, we will define the notion of distance. First, we shall identify the mathematical properties required in clustering. Then we will turn to the way in which the distances can exploit the distribution of the data to be clustered. We will also see that certain metrics are more appropriate in high-dimensional spaces. We shall also see that it is possible to better separate data by expressing the data within another space, thus increasing the contrast between the distances.

A.2.1 Mathematical Properties

Formally, a distance on a set $\boldsymbol{E}$ (in our case, $\boldsymbol{E}$ will be a vector space) is defined as an application:
$\boldsymbol{d} : \mathbf{E} \times \mathbf{E} \rightarrow \mathbb{R}^+$ with the following properties:

- Symmetry: $\forall \mathbf{x}, \mathbf{y} \in \mathbf{E}, \boldsymbol{d}(\mathbf{x}, \mathbf{y}) = \boldsymbol{d}(\mathbf{y}, \mathbf{x})$
- Separation: $\forall \mathbf{x}, \mathbf{y} \in \mathbf{E}, \boldsymbol{d}(\mathbf{x}, \mathbf{y}) = \mathbf{0} \Leftrightarrow \mathbf{x} = \mathbf{y}$
- Triangular inequality: $\forall \mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbf{E},\ \boldsymbol{d}(\mathbf{x}, \mathbf{z}) \leq \boldsymbol{d}(\mathbf{x}, \mathbf{y}) + \boldsymbol{d}(\mathbf{y}, \mathbf{z})$

Examples of distances along with their definitions are presented in Table A.1.

Distances which correspond to this definition alone are not necessarily the best solution for accurately clustering the data. Indeed, such a distance may not necessarily be stable by translation, for example ($\boldsymbol{d}(\mathbf{x} + \mathbf{a}, \mathbf{y} + \mathbf{a}) \neq \boldsymbol{d}(\mathbf{x}, \mathbf{y})$). For this reason, most of the distances used stem from the notion of the norm (the distance is thus expressed by $\boldsymbol{d}(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$), with the addition of the property of homogeneity ($\|\boldsymbol{k}\mathbf{u}\| = |\boldsymbol{k}| * \|\mathbf{u}\|$). Moreover, certain norms derive from a scalar product (by $\|\mathbf{x}\| = \sqrt{\mathbf{x} \cdot \mathbf{x}}$). and therefore have other additional properties. For instance, it is possible to show that the barycenter of N points is the point which minimizes the squares of the distances to these points. This is a very useful property in the field of cluster analysis. The Euclidean distance stems from the L2 norm, itself defined by the usual scalar product. They thus possess all of these properties.

It is nonetheless possible to use functions which do not even meet the criteria of distance definition. This is true for example in the case of Minkowski distances when $\boldsymbol{p} < \mathbf{1}$, which do not satisfy for triangular inequality. It is therefore necessary to check that the chosen algorithm converges without this property.

A.2.2 Data Distribution

Distances such as Euclidean distance or Minkowski distances are independent of the data they are used to compare. However the scales of the dimensions

Manhattan	$\boldsymbol{d(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{n} \lvert x_i - y_i \rvert}$
Euclidean	$\boldsymbol{d(\mathbf{x},\mathbf{y}) = (\sum_{i=1}^{n} (x_i - y_i)^2)^{\frac{1}{2}}}$
Euclidean, Standardized	$\boldsymbol{d(\mathbf{x},\mathbf{y}) = (\sum_{i=1}^{n} \frac{(x_i - y_i)^2}{\sigma_i^2})^{\frac{1}{2}}}$
Minkowski	$\boldsymbol{d(\mathbf{x},\mathbf{y}) = (\sum_{i=1}^{n} \lvert x_i - y_i \rvert^p)^{\frac{1}{p}}, p \geq 1}$
Chebyshev	$\boldsymbol{d(\mathbf{x},\mathbf{y}) = \lim_{p \to \infty} (\sum_{i=1}^{n} \lvert x_i - y_i \rvert^p)^{\frac{1}{p}}}$
Mahalanobis	$\boldsymbol{d(\mathbf{x},\mathbf{y}) = ((\mathbf{x} - \mathbf{y})^T S^{-1} (\mathbf{x} - \mathbf{y}))^{\frac{1}{2}}}$ where $\boldsymbol{S}$ denotes the covariance matrix of the dataset.

Table A.1: Defining Different Distances

are not necessarily comparable. Therefore, when dealing with data relating to a person, for example, the units of age and height are not commensurate. By using the Euclidean distance, data clustering does not discriminate between people according to age alone, because an age difference of one year is as big a difference as a height variation of one meter. The standardized Euclidean distance is much more suited, as when considering each dimension it divides its value by its variance. If we look at this in more detail, using the same example, height and age are correlated dimensions. This correlation carries additional information which can be taken into account. The Mahalanobis distance uses the covariance matrix of the data, thus exploiting the correlation and the variance between the data. However, it is possible for one of the dimensions to be proportional to another. This dimension thus becomes redundant. In such cases, the eigenvalues of the covariance matrix are null, and the Mahalanobis distance therefore cannot be calculated. It is possible to conduct a principal component analysis in order to detect these correlated data and to ignore them. It is interesting to note that the standardized Euclidean distance of the data projected in the eigenspace is equal to the Mahalanobis distance.

Figure A.1 depicts image clustering conducted with a randomly initialized k-means algorithm with three centroids by using various distances. The input data for each pixel is its coordinates in x and y along with its red, green and blue color (thus a 5-dimensional vector space). For each distance function, the algorithm was run three times and the most convincing result (as chosen by a human) was retained. We can see the effectiveness of Mahalanobis distance and standardized Euclidean distance compared with other distances.

A.2.3 The Curse of Dimensionality

When the data originate from a high-dimensional space, we face a problem known as the curse of dimensionality. Dimension reduction is possible by conducting a principal component analysis and retaining only the most significant dimensions. However, this method discards some of the information. A behavioral study of the Minkowski distances on high-dimensional spaces (Aggarwal

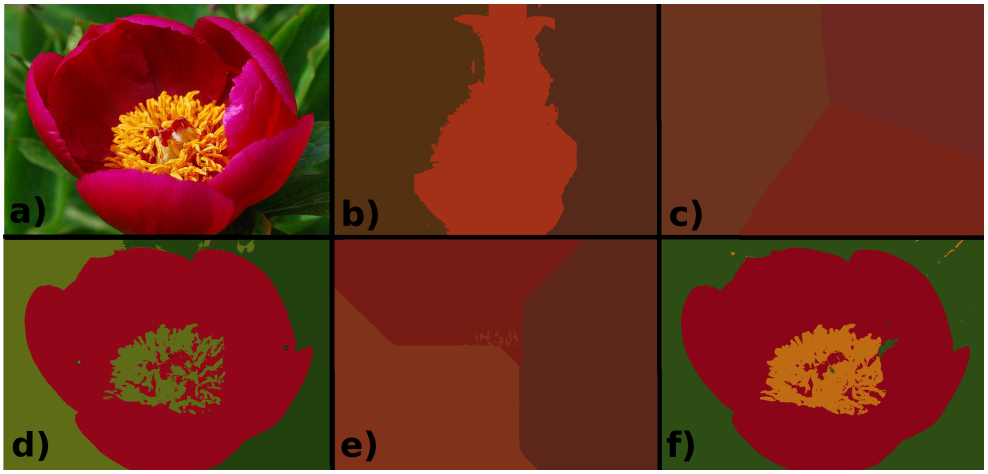


Figure A.1: Illustration of the importance of the distance function on clustering. (a) Image to be clustered, (b) Manhattan Distance, (c) Euclidean Distance, (d) Standardized Euclidean Distance, (e) Minkowski Distance (p=20), (f) Mahalanobis Distance

et al., 2001) shows that the p-distances for a high p-value only exacerbates the problem. As we have already explained, the fractional p-distances are not distances in the formal sense; despite this fact, they can be used to accentuate relative contrast between data. Indeed, they tend to group data together and therefore reduce the curse of dimensionality effect. The Manhattan distance has the advantage of both having triangular inequality and offering better data contrast than Euclidean distance. Furthermore, it is interesting to note in Figure A.1, that the clustering calculated with the Manhattan distance does not divide the image into three equal parts, as in the cases of the Euclidean and Minkowski distances with $\boldsymbol{p = 20}$. The clustering seems better than any regular p-distance (Figure A.1: b., c. and e.). Figure A.2 shows the same image clustered using a fractional p-distance (p=0.2).

A.2.4 Data Separation

Although the curse of dimensionality poses serious problems, processing data with high dimensions also has the advantage that the data are easier to separate. In the majority of cases, N+1 data with N dimensions are linearly separable. In addition, Mercer's theorem (Mercer, 1909) can be used, with a mathematical trick (the kernel trick), to describe the data in a potentially infinite dimensional space. This trick is used especially in classification or regression, particularly in SVMs (Vapnik et al., 1996). However it can also be used to conduct a kernel principal component analysis (Schölkopf et al., 1998). This technique makes it possible to express the data in a higher-dimensional space, in an orthogonal base. Data which are not linearly separable in the initial space become so after being retranscribed in the space created by this

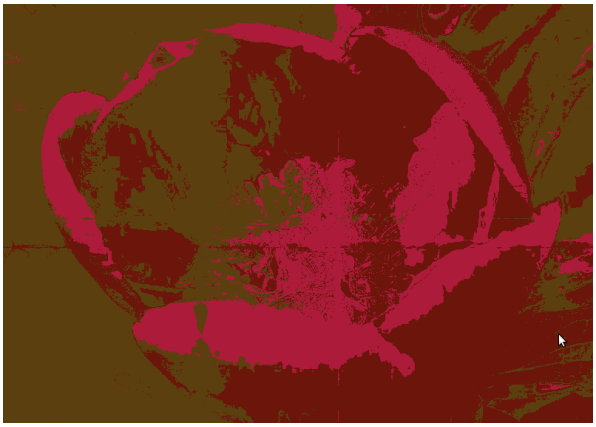


Figure A.2: The flower image clustered using a fractional (0.2) p-distance

	Cluster 1	Cluster 2	Cluster 3
Iris-virginica	31	0	19
Iris-setosa	0	50	0
Iris-versicolor	9	0	41
Iris-virginica	48	0	2
Iris-setosa	0	50	0
Iris-versicolor	4	0	46

Table A.2: Matching matrices for the iris data clustering. Top: with the Mahalanobis distance; Below: with the standardized Euclidean distance after a kernel PCA

technique. The main drawback is that the diagonalization of a $\boldsymbol{M} \times \boldsymbol{M}$ matrix needs to be calculated, where $\boldsymbol{M}$ denotes the number of pieces of data to be clustered. Table A.2 presents clustering conducted by a k-means algorithm on the well-known database of UCI iris data using the standardized Euclidean distance of the data as expressed by a linear PCA and a kernel PCA. A k-means algorithm was used and run 3 times, and the best result is presented in this table.

A.2.5 Summary

We have discussed certain properties of common distances. Table A.3 presents the properties of these distances. Mahalanobis distance seems to be an appropriate choice when the dimension number remains reasonable.

Distance	Property
Manhattan	Relatively good data contrast in high dimensions
Euclidean	The barycenter minimizes the sum of the squares of the distances
Standardized Euclidean	The barycenter minimizes the sum of squares of the distances, Uses part of the data distribution
Minkowski $p < 1$	No triangular inequality, Good data contrast in high dimensions
Mahalanobis	The barycenter minimizes the sum of squares of the distances, Uses data correlation

Table A.3: Summary of the properties of the most common distances

Single linkage	$min(d(x,y)), x \in \mathcal{A}, y \in \mathcal{B}$
Complete linkage	$max(d(x,y)), x \in \mathcal{A}, y \in \mathcal{B}$
UPGMA or Average distance	$\frac{1}{\lvert\mathcal{A}\rvert \cdot \lvert\mathcal{B}\rvert} \sum_{x \in \mathcal{A}} \sum_{y \in \mathcal{B}} d(x,y)$
Average linkage (variation)	$d(\mu_{\mathcal{A}}, \mu_{\mathcal{B}})$ where $\mu_{\mathcal{A}}$ and $\mu_{\mathcal{B}}$ are the arithmetic means of the clusters

Table A.4: Common inter-cluster distances

A.3 Cluster Similarity

Once the notion of similarity between the data is defined, similarity of data in one cluster (intra-cluster similarity) and similarity between clusters (inter-cluster similarity) must also be clarified. Tables A.4 and A.5 present the most commonly used inter/intra-cluster distances. Indeed, these metrics are used by algorithms such as hierarchical clustering. Ascending (or agglomerative) hierarchical clustering iteratively groups together clusters with the greatest similarity (inter-cluster similarity). The result of the clustering is strongly influenced by the choice of this metric. But these metrics also serve to evaluate clustering quality. This evaluation can be used as a stopping criteria, or to choose the parameters of the chosen algorithm (such as the number of clusters for a k-means algorithm for example). In this section, we discuss the impact the choice of metric can have on clustering.

A.3.1 Outlying Data

In most real problems, the dataset includes outliers. They can be caused by a defective sensor or a typing error for example. The presence of such data,

Radius	$\boldsymbol{max(d(x, \mu_{\mathcal{A}})}$ where $\boldsymbol{\mu_{\mathcal{A}}}$ is the arithmetic mean of $\boldsymbol{\mathcal{A}}$
Radius (variation)	$\frac{1}{\lvert\mathcal{A}\rvert} \sum_{x \in \mathcal{A}} \boldsymbol{d(x, \mu_{\mathcal{A}})}$ where $\boldsymbol{\mu_{\mathcal{A}}}$ is the arithmetic mean of $\boldsymbol{\mathcal{A}}$
Diameter	$\boldsymbol{max(d(x, y)), x \in \mathcal{A}, y \in \mathcal{A}, x \neq y}$
Diameter (variation)	$\frac{1}{\lvert\mathcal{A}\rvert \cdot (\lvert\mathcal{A}\rvert - 1)} \sum_{x \in \mathcal{A}} \sum_{y \in \mathcal{A}} \boldsymbol{d(x, y)}$

Table A.5: Common intra-cluster distances

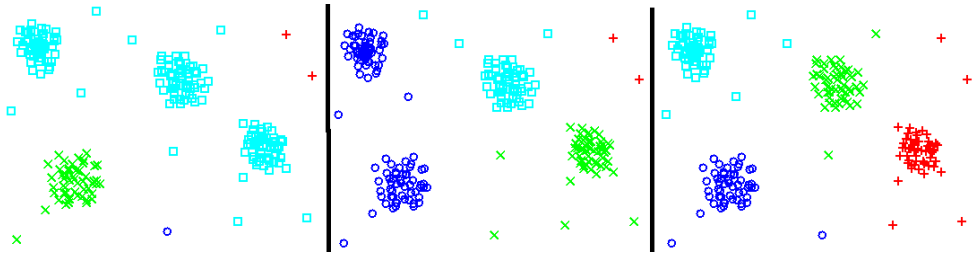


Figure A.3: Comparison of common inter-cluster distance for hierarchical clustering on a dataset containing outlying data. The first column corresponds to single linkage, the second to complete linkage and the third to average linkage

even if there are very few, can greatly influence the inter- and intra-cluster distances. Figure A.3 illustrates these variations, with clustering conducted using SLINK (Sibson, 1973), CLINK (Defays, 1977) and UPGMA. The first row shows that defining the distance by the arithmetic mean of distances between the data of the two groups is much more robust than using the minimum distances of the closest data (SLINK) or the furthest data (CLINK). Indeed, outliers have a tendency to increase intra-cluster distances and decrease inter-cluster distances.

A.3.2 Cluster Shapes

Intra- and inter-cluster distances also influence the shapes of clusters. Of course, the definition of a radius or a diameter for a cluster implies that the cluster is spherical. This hypothesis can be satisfactory for certain problems but not for every case. Figure A.4 illustrates the influence of inter-cluster distance of hierarchical data clustering with clusters of various shapes. In this example, only SLINK calculates the clusters satisfactorily.

A.3.3 Size and Density of Heterogeneous Clusters

Figure A.5 illustrates the influence of inter-cluster distance of hierarchical clustering of data with clusters of various sizes. It can be seen that CLINK has difficulty clustering this type of data.

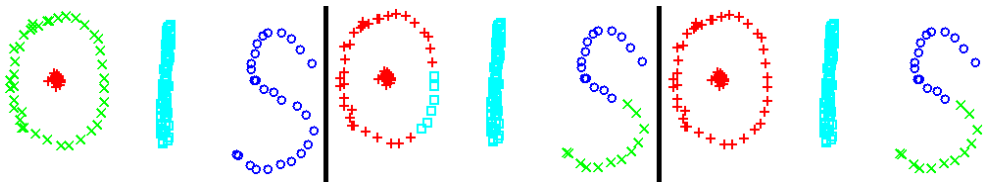


Figure A.4: Comparison of common inter-cluster distance for hierarchical clustering with clusters of various shapes. The first column corresponds to single linkage, the second to complete linkage and the third to average linkage

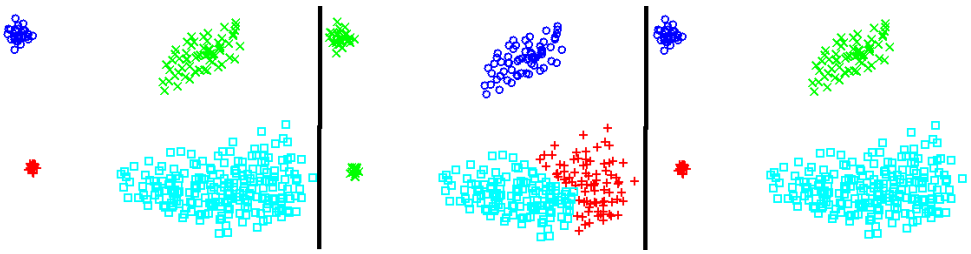


Figure A.5: Comparison of common inter-cluster distance for hierarchical clustering with clusters of various sizes. The first column corresponds to single linkage, the second to complete linkage and the third to average linkage

The CHAMELEON (Karypis et al., 1999) algorithm offers a measurement of cluster similarity which better accounts for the individuality of the data. It consists to construct a k-nearest neighbors graph of the data and uses the notions of relative inter connectivity and relative closeness of two clusters. These two aspects are defined as functions of the clusters's internal connections and connections between two clusters in the KNN graph. This makes it possible to account for data size and density, for example in order not to systematically group together the small low-density clusters into large, dense clusters.

A.4 Temporal Series Similarity

Temporal data analysis is important in fields as diverse as economics, meteorology, speech recognition, movement recognition or even the recognition of handwriting. In this section, we identify the principal difficulties involved by mesuring similarities between time series and discuss about some solutions proposed to handle them during last past decennies.

A.4.1 Time Wraping

A naive approach to defining distance between two temporal sequences is simply to take the sum of the distances of each piece of constituent data. But in such cases both sequences must be of the same size and the calculated distance is not necessarily relevant. In the field of automatic voice recognition for ex-

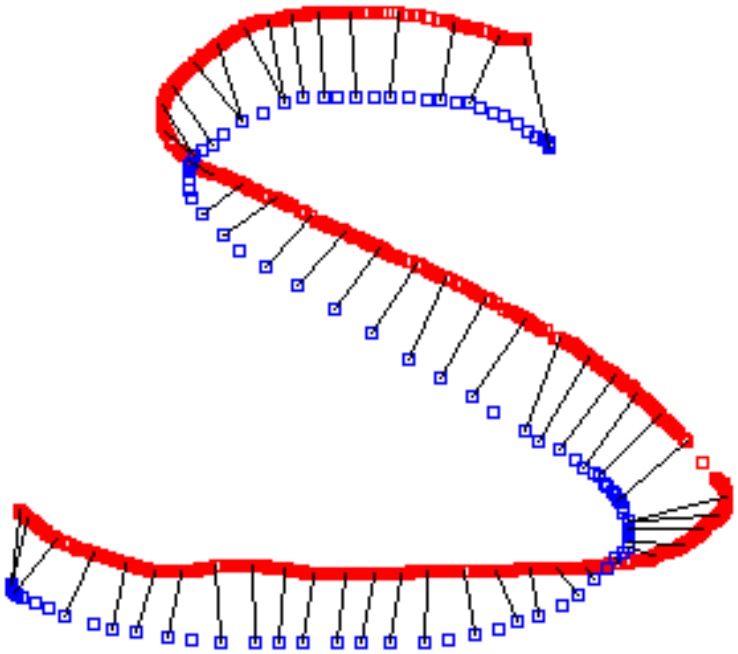


Figure A.6: Alignment between two temporal sequences

ample, the same sentence can be spoken with a completely different temporal dynamic which yields a great difference between sequences even if they represent the same sentence. It is therefore expected that solutions in this field have been suggested, such as Dynamic Time Warping (DTW). This approach aligns the data of each sequence in order to minimize the distance between the data Berndt and Clifford (1994). The distance is then defined as the sum of the distances of aligned data. Figure A.6 depicts an illustration of an alignment between two temporal sequences. Both time and space complexity of this algorithm is $\mathcal{O}(n * m)$ where n and m are the sizes of the two sequences. This can be problematic with large sequences but some less accurate implementations with linear time and space complexity can be used Salvador and Chan (2007).

A.4.2 Translation and Scale

When comparing time series, the information consist in both the data and their positions in the sequence. DTW allows to take into account the relative position of the data within the compared sequences, while comparing absolute

values of the data. When comparing two instances of the same movement made by a person, for example, this movement can be performed with varying amplitude. Two different movements of comparable magnitude provide a smaller DTW distance than two identical movements of different amplitude. This also holds true for position. The distance of the two identical trajectories of the mouse cursor done in a different location on the screen is equal to the distance between two points of the trajectory multiplied by the number of points of the trajectory. However, these differences are insignificant for this type of problem, so it is possible to normalize the data between -1 and 1 Agrawal et al. (1995). Another proposition Bollobás et al. (1997) Das et al. (1997) is to use a linear transformation between the 2 sequences. The key idea is to compute similarity between time series $\boldsymbol{X}$ and $\boldsymbol{Y}$ using a linear function mapping. This approach tries to minimize $\boldsymbol{d(X, \mathcal{F}(Y))}$ where $\boldsymbol{\mathcal{F}}$ is a linear function with $\boldsymbol{f(y) = ax + b}$ for all data within a sequence. Figure A.7 shows a better temporal alignment using this kind of approach than with regular DTW. The distance is also much lower.

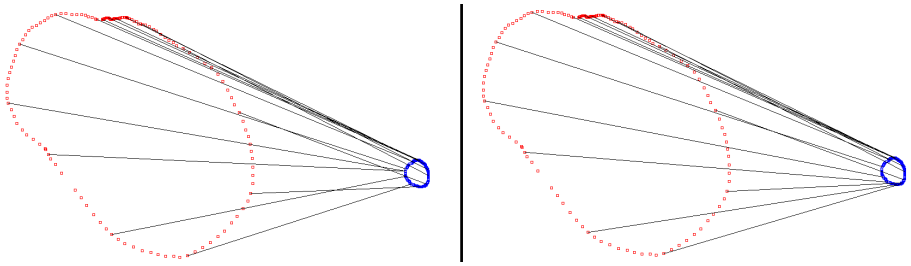


Figure A.7: Left: DTW mapping between 2 time series. Right: translation and scale DTW mapping

A.5 Discussion

In this article, we have discussed the different aspects to be taken into account when choosing metrics for data clustering. We have shown how the properties of the most common distances lead to a better support of many specific problems such as size, density, shape of the clusters, or the curse of dimensionality.

These distances are used in most of the clustering algorithms, whether centroids based algorithms as k-means, neural networks such as self-organizing map (Kohonen, 1982), Growing Cells Structure or Growing neural gas network (Fritzke, 1995a), density based approach as DBSCAN (Ester et al., 1996) or OPTICS (Ankerst et al., 1999), or agglomerative hierarchical algorithms. Again, the choice of the best suited algorithm to the problem is not trivial and a thorough study of the characteristics of these different approaches would be helpful. There is indeed no universal clustering algorithm achieving good quality partitioning whatever the problem. In addition, these algorithms

generally require parameters to be set correctly. Setting these parameters is a particularly difficult problem of data partitioning, because unlike classification problems, we do not have labels indicating the "solution" which would allow one to perform a cross-validation to adjust settings.

Finally, the difficulty to find the main specificities of a given problem (size, density, linear separation of clusters etc.) raises the issue already extensively discussed on visualization of high-dimensional data.

Contents of Chapter B

Appendix B

Chameleon Model IO

B.1 Inputs

B.1.1 High Level Stimuli

CurrentWeaponAmmo	NONE (<10%) LOW(>10%) MEDIUM(>40%) HIGH(>70%)
CurrentWeapon	SHIELD GUN ASSAULT RIFLE BIO RIFLE ONS MINE LAYER SHOCK RIFLE MINIGUN LINK GUN FLAK CANNON ONS GRENADE LAUNCHER ROCKET LAUNCHER ONS AVRIL LIGHTNING GUN SNIPER RIFLE REDEEMER
LifeArmor	LOW (<80) MEDIUM (>80) HIGH (>150)
NumberOfEnemies	ZERO ONE TWO THREE OR MORE
TakeDamage	TRUE FALSE

Table B.1: Inputs specifications

B.1.2 Point of Interest

Yaw	FARLEFT (<-7000) LEFT CENTER RIGHT FARRIGHT (>7000)
Pitch	TOP (>2000) CENTER BOTTOM (<-2000)
Distance	CLOSE (<200) MEDIUM (<1500) FAR (>=1500)
YawSpeed	FARLEFT (<-10000) LEFT (<-2000) NONE RIGHT (>2000) FARRIGHT (>1000)
PitchSpeed	TOP (>500) NONE BOTTOM (<-500)

Table B.2: Points of Interest Attributes

B.2 Outputs

B.2.1 Reflective Actions

ChangeWeapon	SHIELD GUN ASSAULT RIFLE BIO RIFLE ONS MINE LAYER SHOCK RIFLE MINIGUN LINK GUN FLAK CANNON ONS GRENADE LAUNCHER ROCKET LAUNCHER ONS AVRIL LIGHTNING GUN SNIPER RIFLE REDEEMER

Table B.3: Points of Interest Attributes

B.2.2 External Actions

Yaw	FARLEFT LEFT CENTER RIGHT FARRIGHT
Pitch	TOP CENTER BOTTOM
Run	NORUN FORWARD BACKWARD
Shoot	STOPFIRE FIRE ALTFIRE
Strafe	NOSTRAFE RIGHT LEFT
Jump	NOJUMP JUMP DOUBLEJUMP

Table B.4: Points of Interest Attributes

Abstract

This thesis aims to design a behavior model for controlling entities in video games.

The needs of the games industry are constantly changing, partially due to increased computing capacity and diversification of platforms to run games. In the field of artificial intelligence, we identified in chapter 1, the different needs of the industry in this area. We believe that the design of a functional and efficient imitation learning solution would cover most of these needs. In chapter 2, we saw that Data mining techniques can be very useful to offer such a solution. However, for now, these techniques are not sufficient to automatically build a complete behavior which would be usable in modern video games. In chapter 3, we propose a generic model to learn behaviors by imitating human players: ORION. This model is composed of two parts, a structural model and a behavioral model. The structural model proposes a data mining framework, abstracting the different methods used in this field of research, to build a general purpose tool with better visualization possibilities than other existing data mining tools. The behavioral model is designed to integrate data mining techniques within a more general architecture. In chapter 4, we illustrate how we used our model by implementing behavior of Pong players and Unreal Tournament 3 using ORION.

However, the proposed solutions are based on a pragmatic approach. Our model is very flexible indeed to allow more traditional behavior designs while the use of advanced learning techniques is also available. In chapter 5, we identify possible improvements, both of our data mining tool and our behavioral model. Our data mining tool can be improved by more finely modeling Data Mining techniques and by offering a more educational approach. Our behavioral model can be improved in a number of ways: automatic identification of homogeneous sequences by a better time series clustering method, performance based learning and even ideally, automatic construction of Behavior Trees.