THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)

Présentée et soutenue par :
Luc Pons
le lundi 7 juillet 2014

Titre :

Self-Tuning of Game Scenarios through Self-Adaptive Multi-Agent Systems

---

École doctorale et discipline ou spécialité :
ED MITT : Domaine STIC : Intelligence Artificielle

Unité de recherche :
Institut de Recherche en Informatique de Toulouse (IRIT)

Directeur(s) de Thèse :

Pierre GLIZE - Ingénieur CNRS
Carole BERNON - Maître de conférences Université Toulouse III

Jury :
Pierre DE LOOR - Professeur École Nationale d'Ingénieurs de Brest (Rapporteur)
Philippe MATHIEU - Professeur Université Lille I (Rapporteur)
Marie-Pierre GLEIZES - Professeur Université Toulouse III (Examinateur)
Gauthier PICARD - Maître-Assistant ENS des Mines de Saint-Étienne (Examinateur)
Davy CAPERA - UPETEC CTO - R&D Manager (Invité)
Michel MOREL - Ingénieur DCNS (Invité)

Luc Pons

AUTO-AJUSTEMENT DE SCÉNARIOS DE JEUX PAR SYSTÈMES MULTI-AGENTS ADAPTATIFS

Directeur de thèse : Pierre Glize, CNRS
Co-Directeur : Carole Bernon, Maître de conférences, UPS

Résumé

Les jeux vidéo modernes deviennent de plus en plus complexes, tant par le nombre de règles qui les composent, que par le nombre d'entités artificielles qui y interagissent. D'un point de vue purement ludique, mais également en ayant des ambitions pédagogiques, les jeux doivent proposer aux joueurs des expériences qui correspondent à leurs niveaux de compétences et à leurs capacités.

La diversité au sein de la population de joueurs rend difficile, voire impossible, de proposer une expérience qui aille à tout un chacun. Différents niveaux et différentes capacités de progression font que différents joueurs ont des besoins distincts. L'adaptation des jeux est proposée comme une solution pour palier ces difficultés.

Cette thèse propose un ensemble de concepts afin que des concepteurs de jeux, ou des experts de différents domaines, puissent exprimer des objectifs pédagogiques ou ludiques, ainsi que des contraintes sur les expériences de jeu. La généricité de ces concepts les rend compatible avec une grande varieté d'application, potentiellement hors du domaine du jeu vidéo.

Conjointement à ces concepts, nous proposons un système multi-agent conçu pour modifier dynamiquement les paramètres d'un moteur de jeu, afin que celui-ci satisfasse les objectifs définis par les experts ou les concepteurs. Le système est composé d'un ensemble d'agents autonomes, qui représentent les concepts du domaine. Ils n'ont qu'une vue locale de leur environnement et ne connaissent pas la fonction globale du système. Ils ne cherchent qu'a résoudre coopérativement les problème locaux qu'ils rencontrent.

De l'organisation des agents émerge la fonctionnalité du système : l'adaptation de l'expérience de jeu menant à la satisfaction des objectifs ainsi qu'au respect des contraintes.

Nous avons conduit plusieurs expériences pour démontrer que le système passe l'échelle, et qu'il est résistant au bruit. Le paradigme avec lequel les objectifs doivent être définis est utilisé dans des contextes variés pour démontrer sa généricité. D'autres applications démontrent que le système est capable d'adapter une expérience du joueur même quand les conditions de jeu évoluent significativement au cours du temps.

Institut de Recherche en Informatique de Toulouse - UMR 5505
Université Paul Sabatier, 118 route de Narbonne, 31062 TOULOUSE cedex 9

Luc Pons

SELF-TUNING OF GAME SCENARIOS THROUGH SELF-ADAPTIVE MULTI-AGENT SYSTEMS

Supervisor : Pierre Glize, CNRS
Co-Supervisor : Carole Bernon, Maître de conférences, UPS

Abstract

Modern video games are getting more and more complex, by exhibiting more and more rules, as well as a growing number of co-existing artificial entities. Whether they only have entertainment objectives, or pedagogical ambitions, they need to provide a game experience that matches the skills and abilities of players.

The diversity among the player population makes it difficult, if not impossible, to propose a single game that may suit everyone needs. Different skills, preferences, and progression abilities make players need different game experiences at different times. Adaptation of the game experience is advocated as a solution to keep it adequate.

This thesis proposes a set a simple concepts in order for domain experts, games designers or others to express pedagogical or entertainment related objectives, as well as constraints on game experiences. By using only elementary concepts, such as measures and parameters, we remain compliant with a large diversity of domains, even outside of the field of video game.

Along with the expression of game requirements, we propose a multi-agent system designed to dynamically modify the various parameters of a game engine, so that the game experience satisfies objectives expressed by experts or designers. The system is composed of a set of autonomous agents representing the domain concepts, that only have a local perception of their environment. They are not aware of the global function of the system, and they only seek to cooperatively solve their local problems.

From the organization of these agents, the functionality of the system as a whole emerges: dynamic adaptation of a game experience to satisfy objectives and constraints.

We conducted several experiments to demonstrate that the proposed system is scalable and noise resilient. The introduced paradigm with which the requirements must be expressed is used in various context to demonstrate its versatility. Other experiments demonstrate that this system is able to effectively adapt the game experience even when the conditions in which the game takes place significantly change over time.

Institut de Recherche en Informatique de Toulouse - UMR 5505
Université Paul Sabatier, 118 route de Narbonne, 31062 TOULOUSE cedex 9

Contents

Introduction

Computational Intelligence and Video Games

Video games are one of the predilection domain of computational intelligence. Since the beginning of artificial intelligence, problems involving computer games have challenged researchers and computer science enthusiasts, making the field progress and experiment new ways of solving problems.

Programs can be designed to replace a human opponent in classical board games, such as chess or checkers. These games can be modeled mathematically, and by using logic, algorithms can be designed to make a program choose the most appropriate moves. This area led to the famous 1997 chess confrontation between Kasparov and IBM Deep Blue.

Other games are designed specifically to be used on a computer, and put the player in an environment featuring enemies and obstacles. To add a playful dimension to the challenges presented to the player, these enemies often have a certain degree of autonomy. This has led to various challenges, involving automata with planning abilities, or path finding algorithms.

To allow games to be played several times by the same persons, procedural content generation techniques have been used. These techniques generated some aspects of the game, such as environment, or enemies in a pseudo-random fashion to add unpredictability in the game.

Another goal of computational intelligence is to alter the game in order to adapt it to the player. Dynamic difficulty adjustment seeks to propose a game experience that will not be boring or frustrating to the player, by changing the behavior of the game depending on what the player does.

The increasing computing power of modern hardware has led to more and more complex games, that now feature a significantly large number of entities, rules, and means of actions for players. Additionally, the ever growing population of players now features a great diversity. From deeply engaged long time players, to casual players, unfamiliar with the technology, any profile can be encountered.

The need for games to handle this diversity, as well as their large complexity make game adaptation one of the great challenges of modern computational intelligence applied to video games.

Contribution

In this thesis, we propose to address the problem of the adaptation of games to players by using the Adaptive Multi-Agent System (AMAS) theory. We propose to design and implement a system that is able to dynamically modify a game experience to make it match the skill level and abilities of a player.

By following the AMAS theory, we do not design the system as a whole, by successively refining the expressed requirements, until reaching the code level. Instead, we analyze the domain to understand what are the constituting parts of the problem. Following these analyses, we create autonomous entities – agents – that represent the problem to be solved.

These agents adopt the natural behavior of the concept they model. The AMAS theory is based on the consideration that interactions between systems can be of three types. Either cooperative, neutral, or antinomic. The theory states that for a system to be functionally adequate, that is, to have the desired behavior from a designer point of view, it needs to have cooperative interactions with its environment. And in order to be in that state, its constituting parts must maintain cooperative interactions with each other.

Easier said than done, since many problems occur between these constituting entities. We must therefore list these problems, referred to as *non-cooperative situations*, and design means to either avoid them, or to solve them when encountered. The satisfying functionality of the adaptation system as a whole emerges from the local interactions of its constituting parts.

The result of this design is a system able to dynamically modify a game experience, so objectives defined by domain experts or game designers are satisfied. We provide a formalism to describe games that is general enough to be applied to a large majority of games, without relying too heavily on specific game concepts. Moreover, the manner we chose to represent games, as well as pedagogical or entertainment objectives are natural enough to be easily manipulated by domain experts or game designers, with no expertise on computational intelligence.

The proposed system is able to handle complexity of modern games, by dealing with large numbers of parameters at once, as well as several competing objectives, without the need of predictive abilities on games, and without anticipating unforeseen side effects of game modifications.

We are able to adapt game experiences to what a players do in the game, and still not make assumptions on how they think, or react to certain events. By not modeling players' characteristics or profile, we avoid reductive assumptions, or problems that may occur when one individual does not act like most players.

Finally, the approach we describe in this thesis is mostly reactive. This means that the system we have designed reacts to what it observes, and does not learn about its environment. This confers it one particular advantage: it can smoothly embrace change. When a sudden and significant change occurs in the game, or in the player's behavior, the game adaptation system is not penalized by previous activities, and simply reacts fast to the current state of the game and player.

All these characteristics are what we believe to make a game adaptation system satisfying. To demonstrate these properties, we go through a series of experiments, by applying the game adaptation system to different games and systems.

We procedurally generate problems to verify the scalability of the system, and its ability to deal with a large number of parameters, in a noisy environment. We apply it to equation-based population level models, that are known to be highly dynamic and unstable, to verify if these aspects prevent our system from working. We also apply it to two different games, that were not designed with our system in mind, to see how it works in actual games. Finally, we describe the work we have done with our industrial partner DNCS, who exposed its requirements in terms of adaptive games, and demonstrate that our approach satisfies them.

Organization of the Manuscript

This thesis is divided into two parts. The first part consists of a preliminary study of the problem we are interested in, as well as related work, and the second part consists of the contribution and its evaluation. Following are the details of each individual chapter.

- The first part of the thesis is dedicated to the context of the study.
  - Chapter 1 presents the context of this study, and explains why computational intelligence still has a contribution to make in the domain of video game adaptation. Objectives of both learning games and entertainment oriented games are analyzed, and solution requirements are expressed.
  - Chapter 2 proposes a review of the existing approaches. Work has been done on the question of game adaptation, and we review it with regards to the requirements expressed in the first chapter. Other approaches not directly applied to video games are analyzed to determine whether or not they could be used to answer our questions.
  - Chapter 3 presents the AMAS theory, on which the contribution of this thesis is based. The theory itself is detailed, and a related methodology is then introduced.
- The second part of the thesis describes the actual contribution.
  - Chapter 4 is the application of the methodology. Analysis of the problem is conducted in a rigorous fashion, with a software engineering oriented approach. Conclusion about the relevance of the AMAS theory is detailed.
  - Chapter 5 presents the design of the system. Being a multi-agent system, each agent type is presented separately, as well as its behavior. Situations that agents have to handle are described, and means for solving conflicts between agents are detailed.
  - Chapter 6 presents various applications of the proposed system, on simulated problems, as well as on actual games. With these various experiments, we explore several aspects of the problem, and demonstrate that our system has all the properties we claimed to be necessary in the first chapter of the thesis.

We conclude the manuscript by summarizing the contributions on several levels, both applicative and scientific, and by presenting several potential improvements and future work ideas.

Part I

Context and Related Work

1 Context

As described in the introduction of the manuscript, the general idea is to apply artificial intelligence techniques not to play games as it is often the case, but instead to help *design* games, to make them better, so they can offer more satisfaction to those playing them.

In this chapter, we first describe *what* are video games, and we see than more than one type of game exist. Then, we define more precisely the objectives we have in order to make them better. We list the difficulties that one may encounter when trying to solve this problem, and finally, we present a summary of our requirements regarding this problematic question.

1.1 Games and Serious Games

Video games have a rather short historical background, but still many definitions can be found, and there is a great variety among them. After a short definition, we detail the different types of games to give the unfamiliar reader an idea about the subject.

1.1.1 Definitions and Classification

In the 2000's, the video game industry topped the movie industry in several European countries, as well as in the United States. It went from a technology enthusiast hobby in the early 1970's, to a mainstream, mass-oriented entertainment business around the end of the twentieth century. It went from students making a questionable use of university's mainframe computers, to highly profitable video game development studios we know nowadays.

These studios see their activity increasing every year, and more and more resources are now necessary to design a state-of-the-art video game. Through the last 40 years, many genres have emerged, and a large diversity can now be observed among the games that are released every year. Some are really simple and set the focus on their appearance or soundtrack, whereas some others involve complex mechanisms, and have the players face more demanding challenges.

Development practices have evolved too. In the beginning, it was mostly programmers coming up with a game idea and implementing it, mostly for fun. When games became more complex, when platforms became more powerful, design and programming started to become two separate activities. Video game development now consists in many different

tasks, each one of them requiring different skills, and focusing on different aspects of a game.

Playing a video game has consisted in many different things over the years, but certain aspects remain true whatever the era, and whatever the genre. From a computer science point of view, we can define video games as software. Such software manages entities in a consistent fashion, usually located in an environment, dealing with their life cycle, interactions, and behaviors if any. Evolution of these entities are constrained by a set of rules that define the game logic. One or several of the entities in the game are under the responsibility of a human: the player. Through them, the player is able to interact with all the entities that compose the video game, according to the predefined rules of the game world. The game may define explicit goals for the user to reach, or it can be designed to give the player enough latitude to determine preferred objectives. Whatever the design choice, the player must use the mechanisms offered by the game to progress toward the chosen objectives. This process constitutes what is called the game experience, or the *gameplay*.

Given that definition, a little more can be said about the different types of games that can be found nowadays. Although complete classifications exist [Apperley, 2006], they are out of the scope of this thesis. A few concepts are described here in order to give to the unfamiliar reader a general idea on the main video games genres.

Action games are one of the most ancient and prominent category. These are games that often require a certain amount of reflex and dexterity to manipulate resources available to the player in order to compete with virtual entities, other players, or just with the environment.

Strategy games have complex mechanisms and highly detailed game rules. They usually offer the player to control a large set of semi-autonomous entities, in order to dominate a military-like battle with other human or artificial players. This task involves resources allocation decisions, multitasking abilities to control large sets of entities, and strategic planning to anticipate the moves and choices of opponents.

Role playing games have the player incarnate an avatar in the game. One major characteristic of this type of game, is the strong emphasis on the storyline. The player's avatar, usually an individual, has a detailed background, along with its own personality and traits. These games feature a complete and non trivial story, that unfolds as the player makes decisions.

Simulation games try to accurately model various aspects of the reality. Whether it is about sports, management or vehicles, the objective for the player is to master all the aspects of the model in order to reach predefined objectives. Depending on the accuracy of the model, these games can provide a highly profitable training (see section 1.1.2).

It is important to note that these are just a few selected definitions to categorize video games. Many other genres and sub-genres exist, as well as games combining two or more of genres in different degrees. However, these are good elementary game concepts to understand the challenges we will address in the following sections.

Another way to look at a game is not to describe its means of interaction or the nature of the mechanisms offered to players, but instead to look at the very purpose of the game. If the most famous games are designed only to entertain players, there sometimes are other

agenda behind the production of certain titles. These are generally referred to as *Serious Games*.

1.1.2 Seriousness in Games

Many times in the past, games have been used to serve purposes other than the sole entertainment. The world famous board game *Monopoly* has been designed to illustrate economical theories, and analogies have been made between chess and military strategy. Still, it is not until 1970 that the term "Serious Game" is introduced by [Abt, 1970]. Abt suggested that connections could be made between strategy and role playing games, and application domains characterized as "serious", such as the military.

Many serious games have been designed since then, but the most widely accepted definition comes from [Zyda, 2005]: "[a serious game is] a mental contest, played with a computer in accordance with specific rules, that used entertainment to further government or corporate training, education, health, public policy, and strategic communication objectives."

Since the 1970's, many games have tried to take benefit from video games to serve various agendas. The most obvious advantage is the ability for a game to capture attention of players, thanks to the entertainment they provide. They also offer safety in the act of practicing tasks that would have disastrous consequences if misconducted in real life. Additionally, the digital nature of video games allows pedagogical teams to make use of information technology, and thus to improve the communication abilities needed in training sessions or to scale up the whole training process.

Alvarez proposed a classification of serious games according to their underlying objectives [Alvarez et al., 2007].

Advertising games, or *advergames*, use entertainment mechanisms to gain the attention of potential consumers to communicate about a brand or a product. A famous example can be found as early as 1983, with *Pepsi Invaders*, a modification of the best-seller action game *Space Invaders*. In that version of the game, instead of shooting aliens, players must shoot at letters, forming the word Pepsi, the main competitor of the commissioner of the game, Coca-Cola.

Educational games are mainly targeted at children. Though their benefits are not necessarily obvious, many games have tried to deliver an educational content in a more appealing and more interactive form than the traditional teaching methods. As an example, we could mention the *Adi* series, which was famous in France in the 1990's.

Awareness campaign have tried to use video games to spread a message. They make a similar use of video game technology as advergames, in the sense that they want to gain more attention from the public by presenting an entertaining experience while expressing their message. Famous examples are food force, in which players need to deal with hunger crisis, or the 1988 french title *Le Sida et nous* (AIDS and us), in which the player must investigate an unknown epidemic.

Training and simulation games are similar to the previously mentioned simulation games, in the sense that they allow the player to interact with a complex model of vari-

ous domains. If the model is accurate enough, it can represent a good potential for training. Examples of simulation used for actual training can be found in various domains, such as aviation, driving, or management and business. These games are different from the typical educational games because they do not only require the player to acquire a certain amount of knowledge, but they highlight the need for actual skills. Players should be able to deal with potentially never-seen-before situations, in which their experience and skills will allow them to act in an appropriate manner. Alvarez characterizes this category of games with the ”serious play” term: a category in which a scenario is coupled with a simulation, or a “video toy”, rather than a video game. The “video toy” refers to a video game which has potentially no predefined objectives.

From a computational perspective, the games related to that last category are the most interesting. Their exhaustiveness and realism implies a greater challenge level in their design, and their rich interaction with players offers enough complexity to be an interesting subject of study. In the following, the biggest part of our interest is going to be on these complex aspects of game design, rather than those involving awareness or advertising.

1.2 Objectives of Games and Design Challenges

Designing a game is by no means an easy task. There are many pitfalls, and many examples of failures exist in the game development world [Nussenbaum, 2004]. The main objective for a designer is to create an engaging and valuable experience, that will cause the game to reach its objectives, whatever they are. Whether we are talking about a purely entertaining game, or a more training oriented game, we will see in this section that there is a tight relation between these two types of games objectives.

When a commercial game is released, it is usually expected to be enjoyed by those who play it. Financial profit, public recognition, or simply the satisfaction of having created a good game are all motivations for designer to create a *fun* game.

Simulation, training and other serious games need to be fun to be played by a large audience, but more importantly, they need to provide an efficient *learning* process.

1.2.1 Common Objective: Fun and Learning

Fun in the context of games has been extensively studied by Raph Koster [Koster, 2005]. In a nutshell, Koster states that fun is “the feedback the brain gives us when we are absorbing patterns for learning purposes”. In other words, the fun comes when new strategies, or new ways are found to reach these objectives. Once a player completely masters the activities that will let him reach chosen objectives, meaning that there are no more patterns to discover, then there is no more fun. The act of playing becomes tedious, which only means that the player has learned what the game had to offer.

Considering this definition, there is a lot of similarities between entertaining games and serious games. Indeed, both of them seek to put the player in a learning process, whether because the objective is for the player to experience the fun caused by the learning process or to have learned something while having fun.

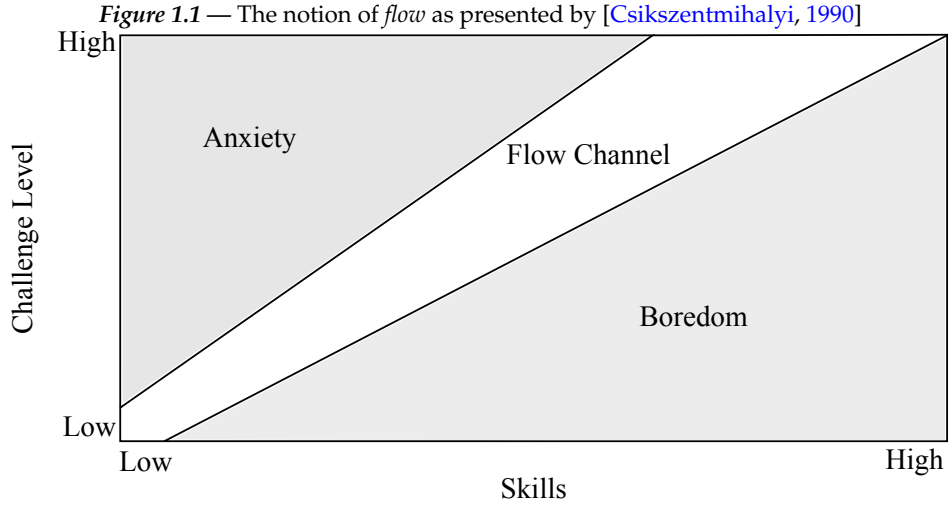
From the player perspective, when a game is getting as fun as it can be, there can be a feeling of "being in the zone". This concept has been studied and formalized in the field of psychology by Csikszentmihalyi [Csikszentmihalyi, 1990]. He defined *the flow* as a "state of mind when consciousness is harmoniously ordered, and [one] want[s] to pursue whatever [one is] doing for its own sake". When one is experiencing the flow, he is completely focused on his activity, with a high level of enjoyment and fulfillment. According to Jesse Schell [Schell, 2008], the flow "is exactly the feeling [game designers] want the players of [their] game to enjoy."

Csikszentmihalyi has given general ideas on how to reach a state of flow, and Shell states that there are four key requirements for an activity to put a video game player into a flow state. Players should:

- have identified clear goals to pursue,
- be able to remain focused on their tasks and not be interrupted,
- receive regular feedback in order to judge the relevancy of their actions, and finally,
- be facing continuously challenging tasks.

If these four points are relevant, the last point is usually emphasized as the main difficulty for game designers [Murphy, 2011]. For that reason, the flow is often illustrated with figures similar to figure 1.1, where it is shown that the challenge level should correspond to the abilities and skills of a player. If a player has more skills than necessary for an experienced challenge, this may result in boredom, and by contrast, if the challenge is too high, it may provoke anxiety.

Figure 1.1 — The notion of *flow* as presented by [Csikszentmihalyi, 1990]



It is interesting to note that theories in the field of education sciences have came up with similar patterns. Even before the notion of flow was introduced, Vigotsky has developed a theory in which he defined the *Zone of Proximal Development* (ZPD) [Vygotski and Cole, 1978]. The ZPD is defined as "the distance between the actual developmental level as determined by independent problem solving and the level of potential development as

determined through problem solving under adult guidance, or in collaboration with more capable peers". In other words, it represents the zone between what a learner can already do, and what that learner would be able to do with some help. The zone therefore represents the interval in which adequate challenges are. Picking a challenge outside of this zone would be counter-productive, as it would be unattainable, or useless. The ZPD is often represented with figures similar to figure 1.1, by replacing the term "flow" with the term ZPD.

These different works enforce the idea that a main requirement for game designers, whatever the game genre they are into, is to offer a challenge level that matches players' skills and abilities.

1.2.2 Pitfalls

Providing adapted challenges to players is easier said that done and designers are likely to encounter major difficulties when trying to reach this objective.

First there is the nature of the population that plays a game. This is especially true in commercial games, and remains relevant in training and simulation games. If the player population is large, there can be a significant diversity in it. A single challenge may be considered differently from a player to another, as players can have different skill levels. What is hard for someone may be trivial for someone else.

In order to compensate these differences among the population of players, a common solution that has been widely used in the commercial video games is to define preset levels of difficulty. They are often referred to as *easy*, *medium* and *hard*. When a player is given the choice to select a given difficulty level, the game experience can be adjusted in order to match that player's abilities. An experimented player can choose a simplified version of the game, with less challenging activities, whereas a more skilled player can avoid boredom by selecting a more challenging configuration for the game, and still face interesting challenges.

If these preset levels can be of some help to put a larger amount of players in conditions which will let them have a fun experience, these levels are generally not enough. If there is a large diversity among the population, two or three levels of preset difficulty may not be enough, and some players may need intermediate levels.

Additionally, it can be stated that skills are not a simple measure that can be valuated on a one dimension scale. The term *skills* can represent many different capabilities, and those involved depend on the domain of the game, and on the nature of the challenges that are dealt with by the player. The lack of precise definition of what difficulty is for a given challenge makes it even harder to design appropriate difficulty levels for a large population.

Finally, when considering a large population of players, learning rates should not be neglected. If, by chance, two players' needs are strictly equivalent at some point, that is, if they have the exact same skill level, and they are able to deal with challenges in an equivalent manner, they may still have different learning rates. This means that one may be able to learn and to progress faster than the other. Therefore, even though their needs are the same, it may not remain true in the future.

Consequently, the variability of players abilities over time may prevent the game from offering constantly adapted challenges even if a game has well designed preset levels. For these reasons, alternatives to preset levels of difficulty are considered to be a potential solution. Next section presents the notion of dynamic difficulty adjustment, and discusses the challenges that come with it.

1.3 Dynamic Challenges Adaptation

Given that preset difficulty levels are not enough to provide adapted game experience to a large amount of players, other mechanisms have to be considered. If it is not possible to determine the best game settings for all the players, then the game settings have to be dynamically changed to match the players' abilities, that is, the game needs to adapt itself to the players.

Adapting the game experience to a player is an interesting alternative to tackle the problem of the adequacy of the game experience. It consists in changing the content of the game as it is played, so the challenge level is always adapted to the players' skills and abilities. By being constantly adapted, the game experience can be at the right difficulty level, providing proper conditions for players, and increasing the chances for players to experience the flow. In other words, it makes a better game.

Raph Koster said that "The holy grail of game design is to make a game where the challenges are never ending, the skills required are varied, and the difficulty curve is perfect and adjusts itself to exactly our skill level." [Koster, 2005]

A game satisfying this property would offer optimal game sessions to players so they can learn what the game has to offer, whether it is serious, or just fun.

If game content adaptation seems like a satisfactory solution, there are some aspects of the problem that make it a complex and difficult problem.

1.3.1 The Problem with People

As mentioned in section 1.2.2, there can be a large variety of players, and describing their skills can be a complex task. Since skills and needed abilities vary from one game to another, it is not possible to be generic and accurate when describing skills independently of a specific game. Moreover, knowledge on players' skills may not be enough to deal with dynamic game modification.

Modification on the game experience as it is taking place can be counter-productive if it does not respect certain rules. What is happening in the game is, in a way or in another, perceived by players. Players then build in their mind a model of what is happening, have expectations on how the game should be evolving, and use these representations to play. If a modification of the game experience ruins these models, it may ruin the believability of the game, and by there ruin the interest the players had for the game. It is important to preserve the believability when modifying a game experience.

Human beings are quite complex, and the processes happening in the brain are still

unclear to the scientific community, even though psychologists understand them better and better. Moreover, humans are highly dynamic entities, evolving dramatically over time in potentially unexpected ways. Therefore, any assumption on how a player will react in any given situation is a risky bet.

1.3.2 The Problem with Games

Koster then states that "In today's world, many of the lessons we might want to teach might require highly complex environment and many moving parts."

This quote is obviously especially true in the case of training and simulation games, since they present an interest when simulating complex environments. But this quote is also relevant with modern entertaining games. With the increasing computing power available in the latest video game platforms (consoles or PCs), there is a significant increase in the complexity of modern games.

The problem that arises when using a game with a high level of complexity is that it becomes difficult to predict anything. A modification of a given aspect of the gameplay may have unforeseen consequences on some other aspect of the game, and nonlinearities in the game system may let loopholes in the design undiscovered. Making strict assumptions on how the game system is reacting to untested modifications may lead to unexpected and costly situations.

All these difficulties make us express a set of requirements that a difficulty adaptation system should satisfy in order to be applied efficiently in the context we have introduced. Next section summarizes such requirements.

1.4 Desiderata for a Game Adaptation System

Dynamically adapting the game experience is considered to be the best candidate solution for making better and more efficient video games. It is however not a trivial problem, and several potentially problematic aspects have been identified in the previous section.

In the following, we describe desiderata for a system designed to provide adapted game experience to players. We provide wished properties for such a system, as well as limits and other characteristics.

Genericity. For the sake of progress in the field, we express the wish for a game adaptation system which could be applied on virtually any game which could benefit from adaptation mechanisms. Of course, there may be exceptions, with appearance of video games of a complete different type, but the point here is to state that an adaptation system should not heavily rely on properties of a specific game genre.

Ease of instantiation. Along with the requirement of genericity, there is a need for the system to be easy to use. The system purpose is to be used by domain experts, that is, pedagogical experts or professional game designers. The experts are the ones who possess

the knowledge of how the game is supposed to be and how adaptation can be done. They are the ones who can judge the overall believability of a game experience (aside from the players themselves). Therefore, they should have means of expressing this information in a convenient way. Additionally, they should not be expected to be computer science experts, in the sense that they should not be expected to know a programming language or how to make extensive use of logics.

Complexity handling. A game adaptation system should be able to work on any kind of game. This requirement has been made explicit already. Since modern games and training simulations can exhibit a high degree of complexity, with nonlinearities, unforeseen dynamics, and lack of predictability, we express a requirement that an acceptable system should not rely on prediction of what happens in the game system. In other words, an acceptable system should be able to work even if no game model is available, and the complexity cannot be apprehended in a straightforward manner: dynamics of the game may lead to delayed consequences of the taken actions, and randomness or noise can be observed since the game platform is in constant interaction with a human player.

No strong assumptions on users. Similarly to the previously expressed requirement, an acceptable system should be able to work regardless of the genre or the application domain. Therefore, if the domain is a niche, with only so few experts in the domain, and so few analysis on the required skills and abilities for a player to be good, the system should nonetheless be applicable.

Additionally, in order to be able to deal with a large variety of players, we express the requirement that no assumptions are made on how the players think, are, or what their relevant skills are. In other words, we want a game adaptation system that makes no hypothesis on players, and makes no use of a users model.

Handling of changes in dynamics. Since we argued that no assumptions should be made neither on the game engine nor on the player population, it should be accepted that anything can happen in the game at anytime. Therefore, an acceptable game adaptation system should be able to deal with unexpected problems and changes with no need for a long computation time. For that reason, we state that a game adaptation system should be used in a real time fashion.

1.5 Conclusion

Video games, even though they are common since the late 1970's, already have a rich history. They have evolved with the technology, and consequently have seen deep modifications of the game experience –or gameplay– they offer to players. Many different genres have emerged over time, proposing different game mechanisms, but also having different purposes. From purely entertaining-oriented productions, titles with educational, training or pedagogical objectives have started to emerge under the name of serious games.

Whether serious or entertaining, games seek to provide players a beneficial experience, that is, either a fun gameplay, and/or an effective learning. For that purpose, literature has shown that one key aspect is to have the challenge level match the abilities of the player.

Modern video games became more and more complex, featuring an increasing number of concepts, means of interactions, modeling reality more and more accurately. Additionally, the population of potential players has grown significantly in terms of both size and diversity. For these reasons, traditional means of providing players with adapted challenges are no longer enough to be satisfying.

Dynamic game adaptation appears as a good alternative to provide efficient games. Dynamically adapting the content of the game experience enables to provide adequate challenges for a large variety of players, whatever their initial skill level or their progression over time.

The variety of games, their growing complexity, as well as the diversity of players and our inability to properly understand the way these players think and behave imply several requirements on a system designed to adapt a game experience to keep it adequate. Most notably, these challenges include the independence of specific game concepts, as well as the lack of strong assumptions or predictions on the behavior of human players. Additionally, the potential inability of providing a comprehensive game model implies that no predictive abilities on a game experience should be needed, and thus, the adaptation process should be able to take place in real time.

In the next chapter, we review a set of works found in the literature that address the problem of game adaptation. Approaches from different fields are analyzed through the criteria we have defined, and a general analysis is then conducted to determine the state of the art.

2 State of the Art

In order to make a survey on existing approaches to tackle the problem of game adaptation to players, different fields need to be analyzed. Indeed, the question can be considered in terms of machine learning, design, optimization or artificial intelligence.

Therefore directly comparing different works may not be an accurate measure of the quality and satisfaction of the different approaches. An abstract analysis on the other hand, conducted using the criteria introduced in the previous chapter can be of a great help to determine the pros and cons of each contribution.

In this chapter, four main parts describe a set of selected approaches. We start with machine learning techniques, and then discuss specific design methodologies. Control and optimization works are then analyzed in the context of our study, before discussing more recent techniques inspired by distributed artificial intelligence.

2.1 Machine Learning for Game Adaptation

A large part of the academic community interested in games and serious games adaptation to the player is focused on artificial entities – or agents – that populate games environments. The main idea, exposed in many approaches, is that it is the behavior of these agents that determine the difficulty level of a game. Therefore, adapting the challenge to a given player consists in modifying the behavior of these agents.

The problem is transformed into a learning problem. How can the agents determine what to do in any condition they face, without the intervention of a human designer?

Several works have addressed this problem in different ways. In the following sections, we detail the most relevant ones, and analyze their results with the criteria introduced in the previous chapter.

2.1.1 Q-Learning

In the machine learning field, the environment in which an entity evolves can be represented as a Markov Decision Process (MDP). A MDP defines a set of states S. A state represents all relevant information for the problem one is interested in, it should be self-sufficient and should not vary depending on previously visited states. In each state, there is a finite set of available actions A from which the artificial entity can choose. An action $a \in A$ makes

the environment go from a state s to another state s', and a reward r is given to this choice by the environment [Otterlo and Wiering, 2012]. The objective of reinforcement learning techniques is to make entities choose the best actions in every state to maximize the reward received from the environment.

Q-learning is an example of reinforcement learning algorithm that makes use of these principles. Its objective is to associate, with each state-action pair of the environment, a Q-value that depends on both the estimated reward from the environment for the chosen action, and the utility of the target state (Q stands for *Quality*) [Sutton and Barto, 1998]. The utility of a state is based on the maximum Q-function one can expect with the optimal policy starting from that state. It is a recursive definition.

More formally, in a Markov Decision Process, Q-learning defines a Q-function:

$$Q(s,a) : S, A \rightarrow \mathbb{R}$$

which is defined as:

$$Q(s,a) = r(s,a) + \gamma max_{a'}(Q(s',a'))$$

- $r(s,a)$ being the reward given by the environment when action a is chosen when the process in the state s.
- γ is a parameter that defines the importance of the immediate reward with regards to the utility of the action.
- s' is the target state of the action a
- $max_{a'}(Q(s',a'))$ is therefore the maximal $Q - value$, for the best available action a' when the environment is in the target state s'.

The Q-learning algorithm seeks to compute all Q-values for all actions in all states of the environment. For that purpose, the Q-values of all state-action pairs are initialized, and then the MDP graph is explored. Thanks to the reward given by the environment, the Q-values are updated with the following rule:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \times [r(s,a) + \gamma max_{a'} Q_t(s',a') - Q_t(s,a)]$$

α being a learning rate parameter that defines the importance of the newly observed value with regards to the previously memorized one.

Stutton and Barto proved that the algorithm eventually converges to appropriate Q-values. However, it may need an indefinite number of iterations. Q-learning has been applied in several domains, including adaptation of video games.

[Andrade et al., 2004] used Q-learning in the context of fighting games, to make an artificial opponent choose its actions depending on the state of the game. A fighting game is a game in which two players are confronted to each other, and their objective is to defeat their opponent. In this case, a human player is confronted to a virtual entity –a bot– that is supposed to adapt its behavior to provide an adapted challenge to the human player.

A state representation is introduced, encoding various gameplay information, such as the distance between players, or their respective state (attacking, defending). Actions are deduced from the gameplay, that is, attacking, defending, moving away or getting close to the opponent. The reward of each state is defined as the health difference between players, since it is a relevant indication of how well the artificial entity is doing: a player with a health higher than that of its opponent gets a positive reward.

The game state-space needs to be explored a great number of times in order for the Q-values to be accurate. Quoting Andrade et al.: "It could not be possible to learn a competitive behavior in real-time. To deal with this, we use an offline learning phase, where a general initial behavior is learned by the agent." The game state-space is therefore manually explored before a player can actually be confronted to the artificial player.

Once done, the artificial player is able, for each state, to choose the action that will lead to the greatest reward, in other words, to be as efficient as possible. However, for an adapted game experience, one does not necessarily need a highly competitive player, but instead, a challenge level that matches one's skills. For that purpose, artificial players will not always select the action with the highest Q-value, but instead, will select the second action, or the third action, until the ranking of the actions leads the challenge to be adapted. By contrast, if the game is too easy, the range of the selected action is increased until the game is balanced again. In this contribution, a balanced game is defined by a *challenge function*, which basically defines that games are balanced if the virtual player inflicts as much damage as it receives.

Table 2.1 shows a summary of the analysis of Q-learning. The approach only relies on the modification of Non-Playing Characters' behaviors (NPCs), and is thus limited to the type of games where the impact of modifications on such behaviors is significant. Moreover, the adaptation mechanism relies entirely on the offline learning phase, where the game state-space needs to be explored completely several times to determine the value and utility of each action in each situation. Besides, all aspects of the gameplay need to be included into a consistent model in order to (1) be able to propose all possible actions, and (2) to properly determine the value of the reward. If this is possible in a simplified one-versus-one fighting game, it may not be realistic with the majority of the other games, that generally present a higher level of complexity.

Table 2.1 — Q-Learning adequacy to the defined criteria for the game adaptation system

Characteristic	Adequacy	Comment
Genericity	−−	Relies on the existence and importance of NPCs
Complexity handling	−−	Needs an exhaustive description of the game states and actions
No strong assumptions on users	++	No user modeling required
Handle unforeseen situations	−−	Cannot afford change of dynamics

2.1.2 Dyna-Q

Because the learning phase takes a significant amount of time, and cannot be expected to have satisfactory results in real-time, enhancements of Q-learning have been proposed. Using a model of the environment in the algorithm can speed up the process of learning an appropriate policy. It is for instance the case of the *dyna-Q* algorithm. This algorithm is similar to the previously described Q-learning algorithm, but at each iteration, in addition to estimate the Q-value of each state-action pair, the algorithm builds a model of the evolution of the process.

More precisely, each time an action a is chosen, leading from a state s to a state s' the model is updated to remember this new information. Similarly, the average reward given by the environment for having chosen action a in state s is updated. Once the Q-value of s' is updated, similarly to regular Q-learning, a planning phase occurs. The planning phase consists in choosing random previously memorized state-action pairs from the model, as well as target states and received rewards, and in updating the Q-values of the target states accordingly.

This mechanism speeds up the first phases of the algorithm, because state-action pairs can be memorized long before the recursive definition reaches them.

[Amato and Shani, 2010] have applied Q-learning and Dyna-Q to the game adaptation problem. Similarly to the previously described work, they chose to modify the game difficulty by changing the behavior of computer controlled opponents. Their work takes place in the context of strategy games, and more specifically the game *Civilization*[1]. In such games, players are confronted to each other. There are many different aspects in the game, and their means of interactions are broad. Players need to manage units, cities, and military. As authors state "The large state space and action set, uncertainty about game conditions as well as multiple cooperative and competitive agents make strategy games realistic and challenging".

They consider a player facing a single opponent. They also consider four preexisting high-level strategies that artificial opponents can use to face human players, which can be differentiated by characteristics such as aggressiveness, or diplomatic skills. The adaptation mechanism in this work consists in learning which strategy an artificial opponent should adopt in a given situation. This greatly simplifies the problem, and no further concerns are expressed regarding the previously mentioned dynamical and uncertain aspects.

The state space is defined through a set of four different parameters, consisting of differences of population, military, land and power between players. Each of these parameters can take three different values. The reduction of the game complexity to these parameters leads to 81 different possible states for the game. The reward model is defined by the difference of score between players at a given time.

For the learning process, it is assumed that the human player will adopt a fixed strategy repeatedly. The algorithm is allowed to learn for 100 games, before it is observed that there is a 15% improvement in the artificial player results. This means that over the course of 500 games, the share of the game won by the learning artificial player is increased by 15%.

[1] http://en.wikipedia.org/wiki/Civilization_IV

Dyna-Q offers improved performance over Q-Learning, but overall, it has the same weaknesses, as listed in table 2.2. The need for an exhaustive description of the game states, as well as the need for the algorithm to encounter the same situations repeatedly make it hard to apply the algorithm when adaptation is required as the player changes.

Table 2.2 — Dyna-Q adequacy to the defined criteria for the game adaptation system

Characteristic	Adequacy	Comment
Genericity	−	Relies on the existence and importance of NPCs
Ease of instantiation	++	The algorithm is usable as is
Deals with great complexity	−−	Needs an exhaustive description of the game states
No strong assumptions on users	++	No user modeling
Handles unforeseen situations	−−	Cannot afford change of dynamics

2.1.3 Learning Classifier Systems

[Holland, 1975] introduced Learning Classifier Systems (LCS), as a means for an artificial system to learn how to solve a given problem. Even though there are a lot of similarities between LCS and other approaches of reinforcement learning, such as Q-Learning, differences in the environment representation give more flexibility. Moreover, the coupling with Genetic Algorithms (see section 2.3.3.1) enables unforeseen situations to be dealt with.

In LCS, the environment of the solving system is represented as a set of properties, that can be satisfied or not. From this representation, a LCS employs a set of rules –*classifiers*–, that associate actions with states of the environment. Rules can be more or less restrictive. The condition part of the rule maps the different attributes characterizing the world to a needed value. The demanded value of each characteristic can be either 1 (satisfied), 0 (unsatisfied) or # (no constraints). The # symbol allows generalization, and removes the necessity to exhaustively describe the environment. Besides, in LCS, not all the possible environment states must be represented by classifiers, since the set of classifiers evolves over time, as described hereafter. The action part of classifiers can be described with an *ad hoc* notation, depending on the domain concepts.

With each of these classifiers, is associated a *strength* value. Similarly to Q-learning, this strength value represents an expected payoff from the environment if the involved rules' action is selected and applied.

At each activation cycle of the LCS, three main steps are involved: performance, reinforcement and generation.

The performance step consists in choosing the action to apply in the environment. When the environment is perceived, the set of classifiers that have conditions that match the current environment are selected to form the *matching set* M. From M, there is a selection mechanism. Usually classifiers with the same actions are grouped together, and the set with higher *strength* values is selected, but there can be variations. The selected set is referred to as the *action set* A, and the action of the classifiers forming A is denoted α.

The objective of the reinforcement step is to update the strengths of the classifiers. Once the action α has been applied, a payoff is received from the environment. This payoff can be used to confirm the strength of the classifier, and potentially the choice of the previously selected classifiers that lead to the current states. Several strategies exist. One of them, described by Holland, is inspired by economical systems. Classifiers get paid if they perform well in the environment, and they need to pay those that helped them get where they are now. More precisely, the payoff, or *immediate reward* R is distributed to each classifier of A. Each classifier of A increases its strength by $R/|A|$. A portion $\beta \cdot strength$ of each classifier of A is collected from them, and redistributed equally to the classifiers of A^{-1}, which is the action set of the previous cycle. This mechanism back-propagates the reward to previous classifiers, and allows the system to maintain consistent chains of actions.

Finally the generation phase aims at providing new rules to enhance the performance of the LCS. There are two majors components of that phase: the covering operator, and the genetic algorithm. The covering happens when no classifier matches the current state of the world, or if all the strengths of selectable classifiers are below a certain threshold ψ, leading to the impossibility to build an action set. ψ is usually a function of the strengths of the total classifier population. In these cases, a new classifier is created corresponding to the current state of the environment. Each characteristic can be generalized, i.e. the # symbol has a given probability to replace each part of the newly created classifier condition part. The action part of the classifier is chosen randomly. In order to maintain the size of the classifier population, one of the weakest classifier is deleted. Genetic Algorithms (as detailed in section 2.3.3.1) allow to produce new solutions to a problem, based on existing solutions. In this case, new classifiers can be produced by combining two of the existing classifiers with the highest strengths. Similarly to the covering algorithm, classifiers with a low strength value are deleted to preserve the size of the population.

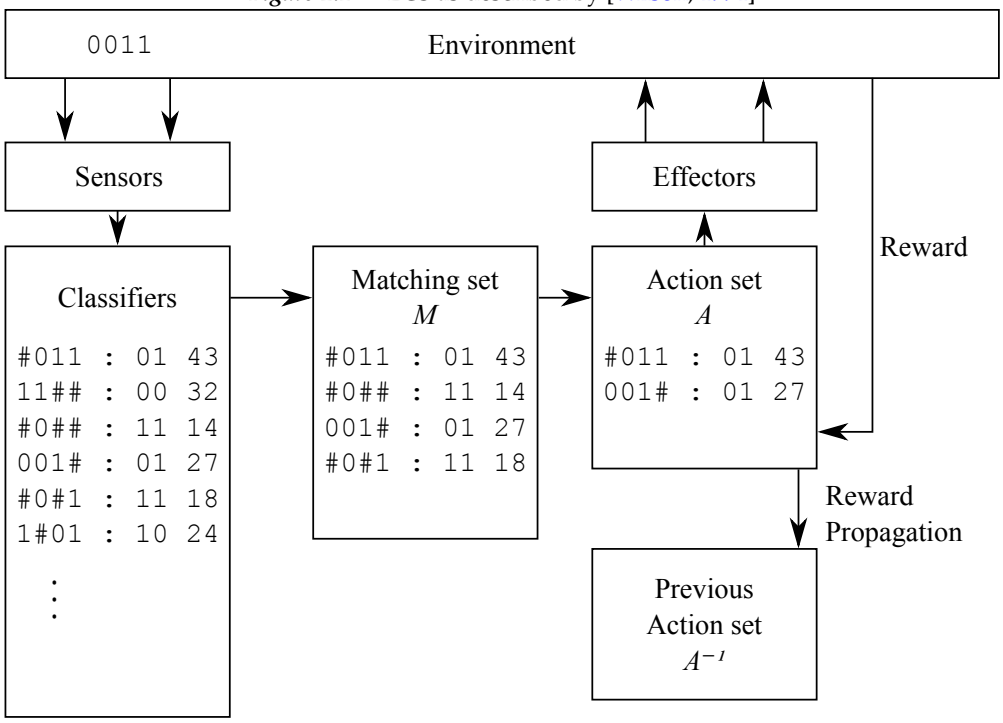
[Tsapanos et al., 2011] have applied this machine learning technique to real-time strategy games. They put an emphasis on two specific aspects of the game: the production of unit, and the actions of military unit. Their objective is to build an artificial player that learns how to play. For that purpose, they design two Learning Classifier Systems, one for each of the mentioned aspects of the game. Even though the first goal is not directly to provide an adapted challenge, the artificial player shows some adaptation abilities, since the optimal strategy to face a human player is not always predictable.

To describe the state of the world, specific variables are selected. They can be numeric (distance, quantity of specific units) or boolean. For each numeric variable, four significant intervals are selected, and assigned to a two-bit combination. This way, the value of the variable is reduced to two bits. For a boolean variable, only one bit is necessary.

The first classifier system is the one in charge of the creation of units. The classifiers condition part is composed of 10 bits. Eight bits are dedicated to the representation of the number of the two types of units the player and the enemy currently possess, one bit is used to express the existence of threat to the player base, and finally one bit represents the availability of resource in the sight of the player. The action part of the classifiers consists only in the creation of one of the two types of units.

The classifiers of the second system, in charge of controlling military units, are a bit more

Figure 2.1 — LCS as described by [Wilson, 1994]



complex. Military units can be doing four different tasks: defending the base, defending resources, exploring, or attacking. Therefore, the condition part of the classifiers uses 16 bits, and represents, in addition to the perceived number of enemy units, the amount of military units that are currently in each of the four states. The action part of the classifiers consists in creating a military unit in one of the four categories.

The Learning Classifier Systems are run to learn optimal policies. In table 2.3 we can see that they rely on the importance of non-playing characters behaviors to modify the game experience, but that their ability to generalize the description of the environment allows them to handle more easily more complex games, since no exhaustive description is required; however, and similarly to the Q-Learning applications, the reward from the environment comes at the end of the game. Since it takes time for the reward to be propagated into the system, and to assign a relevant strength value to classifiers, it takes several games with a static opponent for the LCS to learn how to behave, and to outperform other learning systems.

2.1.4 Classifier Systems in Intelligent Training Systems

Buche et al. have used learning classifiers systems in order to autonomously assist human players and create an Intelligent Tutoring Systems (ITS) [Buche and Querrec, 2011]. An ITS places a human player in a virtual environment, and proposes exercises and challenges. As the players deals with the proposed tasks, the challenges are modified.

In their work, they do not seek to create new classifiers, or to combine existing ones in order to modify the classifier base. In other words, they propose not to use a generation

Table 2.3 — Learning Classifier Systems adequacy to the defined criteria for the game adaptation system

Characteristic	Adequacy	Comment
Genericity	−	Relies on the existence and importance of NPCs
Ease of instantiation	++	Domain concepts are used as inputs
Deals with great complexity	+	Game states are described using generalization
No strong assumptions on users	++	No user modeling
Handles unforeseen situations	−−	Significant time and several iterations are needed to discover new actions to perform

phase in the system life-cycle. Instead, the focus is set on finding appropriate rules weight, so the proper policies are applied.

The classifiers condition parts characterize the pedagogical situations with regards to several models that need to be built by domain and/or pedagogical experts. These models represent the business domain in which the game takes place, the potential mistakes that can be made by a trainee, the state of the interface of the tutoring system (e.g., are given objects currently visible?), and finally the state of the trainee.

The action parts of the classifiers can use any mechanism offered by the platform with which the trainees are interacting. These rules, along with the various needed models have to be written by domain experts, and will not be constructed by the LCS.

Classifiers are organized in three hierarchical levels (see figure 2.2). The first and most abstract level is referred to as the *pedagogical approach*, the intermediate level is the *pedagogical attributes*, and finally the most concrete level is referred to as *pedagogical techniques*.

In each of these levels, classifiers are shared by *pedagogical orientation sets* that represent different approaches, attributes or techniques. Each of these sets can consist of several rules.

At the beginning of the process, arbitrary weights of 1 are associated with the sets of the first level. Each individual rule of this level activates a set of the second level. Each valid rule of the first level transmits the weight of the set it belongs to to the set it activates at the next level (this is represented by thick arrows on figure 2.2).

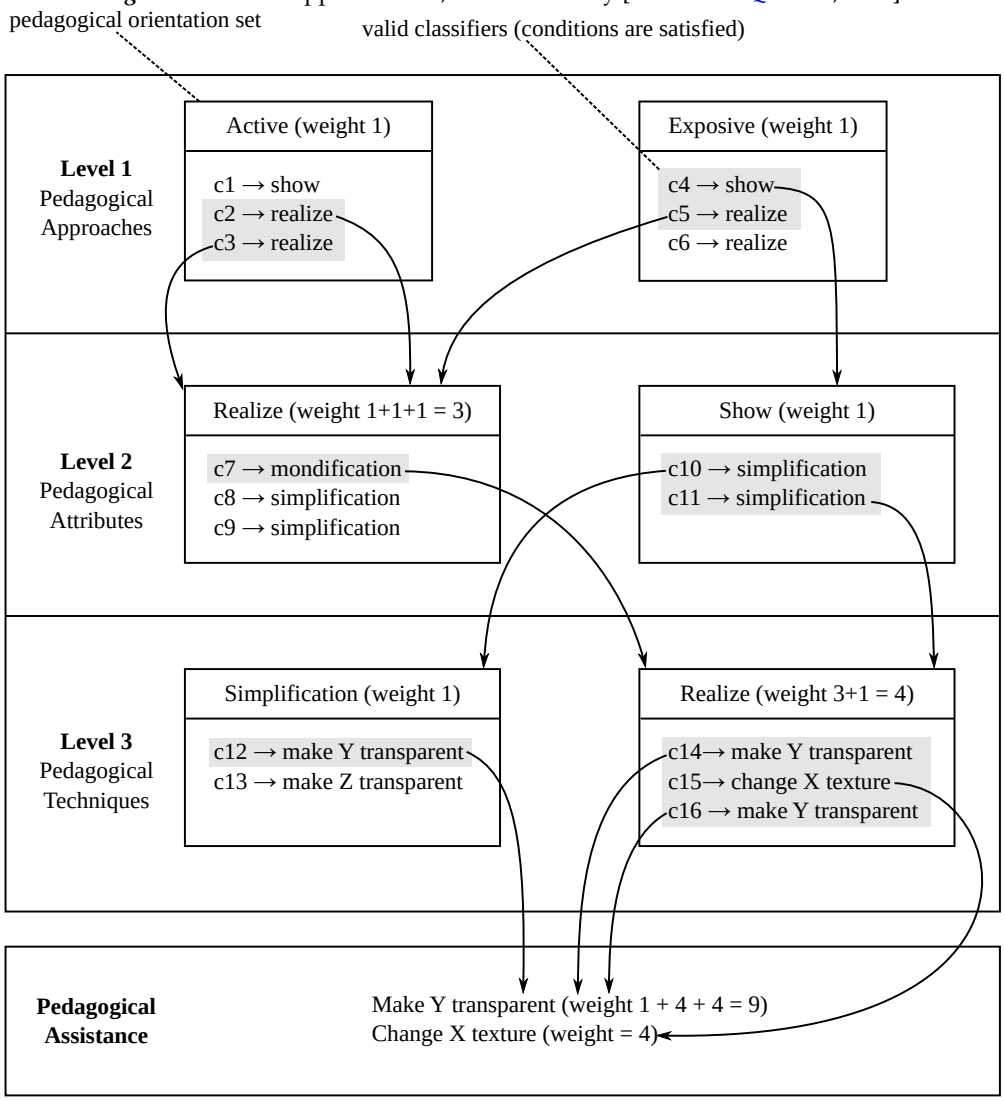
Therefore at the second level, the weights of the sets depend on the weights of the sets of the first level, as well as of the validity of the rules of the first level.

Similar calculations are done between the second and the third levels, as well as between the third level and the actual *pedagogical techniques*. Therefore a weight is computed for each of these *pedagogical techniques*, which are actual actions to be done on the game platform.

These *pedagogical techniques* are sorted thanks to their weight, and are sequentially presented to a human pedagogical expert. This expert can accept or reject the proposals. An accepted proposal gets a positive reward. Similarly to the mechanisms described in the previous section, the reward is back-propagated to previous rules. In this case, to rules of

the second and first levels. The weights of the rules that led to the one that was ultimately chosen by the expert are increased, whereas the rules that led to those that were ultimately rejected are decreased. This mechanisms allows to capture the expertise of human experts in the weighs of the rules.

Figure 2.2 — LCS applied to ITS, as described by [Buche and Querrec, 2011]



After having learned the expertise of human domain experts, the system is able to modify the game platform as a human would while the player is interacting with it. However, we could argue that if a player does something that creates a situation that has never been seen before, then the expertise of the human expert may not be available, as it may have not been learned for this specific situation.

Given the complexity of the game and the amount of potential situations, experts may need to train the system for a great amount of time, with a great diversity of human players.

The diversity of players itself must be captured in models that give all relevant information to the classifier system, which may not be trivial, and may not be reusable from one game to another. Similarly, a game model is needed for the LCS to reason, and again, this

may prevent the approach from being applied in complex or unusual games.

Finally, since the classifier base is fixed, experts need to write all classifiers from scratch, and must not forgive useful rules, which, depending on the complexity of the game, may be a great amount of work. Table 2.4 summarizes these considerations.

Table 2.4 — Learning Classifier Systems applied to ITS adequacy to the defined criteria for the game adaptation system

Characteristic	Adequacy	Comment
Genericity	++	Can be applied to a wide variety of games
Ease of instantiation	−	All adaptation rules need to the manually written
Deals with great complexity	++	Classifiers are powerful enough to describe complex situations
No strong assumptions on users	−−	Heavy usage of user modeling
Handles unforeseen situations	−−	The system must be trained on situations to be efficient

2.1.5 Dynamic Scripting

Dynamic scripting, proposed by [Spronck et al., 2006], also considers games in which a player has to face one or several entities. The challenge level is determined by how these entities are behaving. If they are given appropriate strategies, they may defeat highly skilled players, while if they do not behave in efficient ways, they can be defeated even by beginners. Therefore it is argued that for any given player at a given time, an optimal strategy that fits the needs of the player exists.

Opponents are considered to be autonomous entities, whose behaviors are designed using scripts, as it is often the case in modern video games. A script consists of a set of rules that trigger actions when given states of the world occur. Spronck et. al suggest that a rule base should be established, from which certain rules could be selected to instantiate a particular script. The rule base is simply defined as the set of all rules available for a given entity type. Therefore one rule base for each type of opponent the player will have to face should be defined. Each rule is associated with a weight $W \in [W_{min}, W_{max}]$. When an opponent is instantiated in the game, a script is generated with rules selected from the rule base. The probability that a rule is selected from the rule base to be incorporated into the script is proportional to its weight.

Consequently, the objective of the adaptation process is to dynamically modify the weights of the rules in the base in order for the generated scripts to be more suitable to players.

The authors define the notion of confrontation between artificial opponents and players. They are parts of the game experience in which the player is facing a challenge, and the activity of the artificial entity is preventing, in various degree, from being successful. The outcome is the measure of how well the player did in the challenge. Authors state that confrontations should be identified, and outcomes should be measured. By the end of the

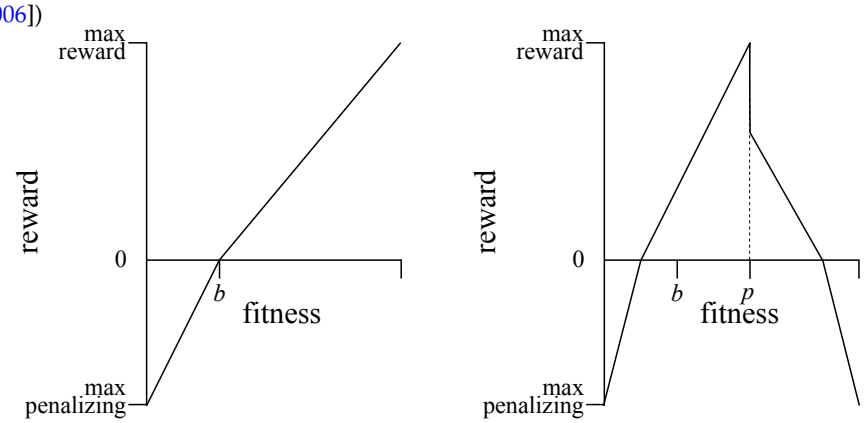
confrontation, a fitness value for the entity the player was confronted to is computed depending on (1) the issue of the confrontation (whether the entity won or lost), (2) whether the entity is alive or not, (3) its remaining health, and finally (4) the damage it inflicted to the player.

If the fitness of the entity is below a *break-even* threshold, the weight of the selected rule is decreased (the lower the fitness, the lower the new weight). If the fitness is higher than the *break-even* value, then the weight of the selected is increased (the higher the fitness, the higher the new weight). Figure 2.3a shows the reward value of a rule according to its fitness value. These rule reordering mechanisms lead to more and more efficient artificial opponents, that maximize the fitness value after each confrontation with the player.

In order for the opponents to be adapted to the players and not simply stronger and stronger, three mechanisms of rule weights update are proposed.

The first mechanism is named *high-fitness penalizing*. It defines a new value p as the reward-peak value. When the fitness reaches p, the reward of the rule is maximal. The reward starts to decrease as the fitness gets higher. Figure 2.3b shows the reward of the rules depending on the fitness value. With this mechanism, an entity does not become stronger and stronger, but instead "the higher p is set, the more effective the opponent behavior will be". Therefore, to adapt the game difficulty as it is played, p is modified after each confrontation between opponents and the player. p is initialized to a low value, it is increased each time the player wins a confrontation and is decreased each time the player looses a confrontation.

Figure 2.3 — Reward attribution to rules according to their fitness (from [Spronck et al., 2006])



a) Regular reward attribution according to the fitness value

b) Reward attribution according to the fitness value with the *high-fitness penalizing*

The second mechanism is referred to as *weight clipping*. It simply consists in defining a limitation (W_{max}) to the weights of the rules. When a rule is efficient, it gets high rewards. Consequently, its weight becomes higher and higher, resulting in a greater chance of being selected in a script. Setting a limitation W_{max} on the weight of a rule means that even if the rule is actually efficient, its weight will not increase beyond W_{max}, and therefore, will not see its selection probability be as high as it could. Therefore suboptimal rules are still selected,

resulting in less efficient artificial opponents. The idea of *weight clipping* is to increase W_{max} when the player wins, and to decrease it when the player looses.

The third mechanism is *top culling*. It is an extension of weight clipping. The difference is that it allows rules weights to exceed the limitation W_{max}. However, rules with a weight greater than W_{max} cannot be selected to instantiate scripts. Therefore, when the opponents are inefficient, W_{max} is high and efficient rules can be selected, whereas when opponents are efficient, W_{max} is high, and rules with a high weight are not selected, resulting in poor strategies.

Table 2.5 shows that dynamic scripting is an efficient approach, as it adapts the game experience as the player evolves over the succession of confrontations with artificial entities. The use of behavior rules is a concept that should be familiar with game designers, and will generate no additional concepts for them to learn. It still however relies on the fact that the game experience is only adapted thanks to the modification of artificial entities behaviors, while other aspects of gameplay are not taken into account. It remains one of the most interesting approaches for game adaptation in the field of machine learning, as it does not necessitate large amount of data, nor many iterations to provide efficient results. If no predictions are made about how the game is going to unfold in the near future, the potential delayed consequences of the technique actions are not taken into account. It is possible that the new weight of a particular rule is actually optimal, but given the nature of the game, this optimality will not be observed before a given amount of time. Since no anticipation of that delay is present in the technique, it is possible that the challenge level oscillates largely around the optimal, by successively being too high and too low.

Table 2.5 — Dynamic Scripting adequacy to the defined criteria for the game adaptation system

Characteristic	Adequacy	Comment
Genericity	−−	Relies on the existence and importance of NPCs, plan-based behaviors are required
Ease of instantiation	++	Input of the algorithm takes the form of behavior rules
Deals with great complexity	++	No modeling of the system is needed. Solving process occurs among distributed entities
No strong assumptions on users	++	No user modeling
Handles unforeseen situations	+	On-line adaptation after each confrontation

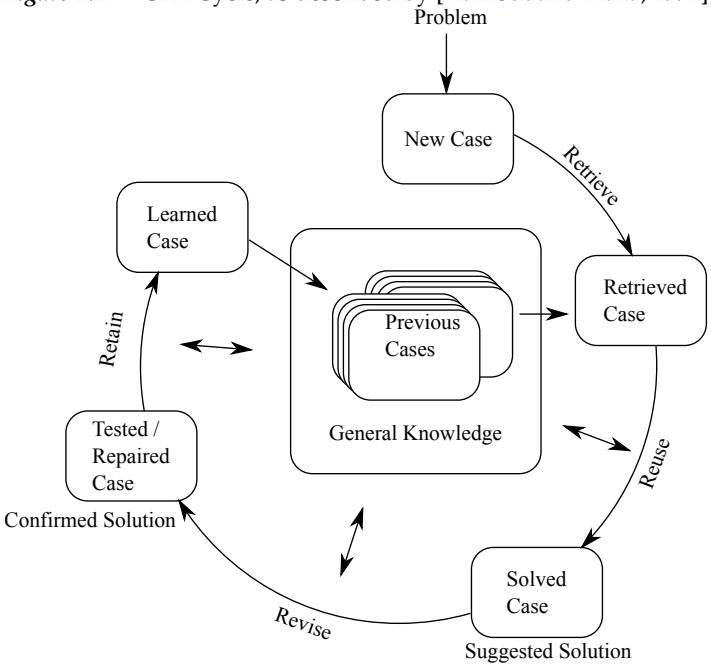
2.1.6 Case-Based Reasoning

To avoid the need for artificial entities to experience all possible situations a large number of times, other approaches have tried to add some generalization capacities in the process of learning. A good example is Case-Based Reasoning (CBR), which has been introduced by [Kolodner, 1992], and has known a significant success since then, both on the academic

world and in industrial applications.

CBR basically consists in remembering previous experiences and in using them to choose what to do in novel situations. Each time a new action is chosen and applied, the observed result is remembered for future uses. CBR is executed as a cycle, where previous cases are reused, applied, and knowledge is updated. It is often argued that CBR mimics the way humans solve problems from experience.

Figure 2.4 — CBR Cycle, as described by [Aamodt and Plaza, 1994]



A common acceptation of the CBR cycle is shown in figure 2.4. The experiences are stored in a base in the form of cases. A case represents an action to apply, associated with a context of the involved system. The representation of the context of a case can be an attribute-value list, but depending on the domain, more complex representations can be designed (graphs, object-oriented, and so on).

When a novel situation is encountered, the CBR engine seeks to *retrieve* the previously stored situation that is the most similar to the current one. A similarity measure is therefore needed to evaluate the proximity of cases. With attribute-value lists, the similarity could be the means of similarities of all attributes, but different functions can be used, depending on the importance of each attribute. Basically, the domain experts must provide a similarity function close to the effective reusability of solutions, and it should also be easy to compute.

Once the most similar case has been retrieved from the case base, the corresponding action is applied to the current situation; the system *reuses* it. The action could be applied as is, or, depending on the domain, it may need to be adapted to the current situation. Adaptation could be an automatic parametrization of the action or a more complex rule-based transformation system. Depending on the domain, it could be possible to use a manual adaptation of the solution. This is called the *revise* phase.

Once the solution is effectively applied to the real system, its effectiveness is evaluated.

Depending on its quality, the solution can be modified or confirmed. The *retain* phase memorizes in the case base the newly generated information. Given the output of the *revise* phase, new cases can be created, or old solutions can be modified. This phase may also allow the designer to update the similarity function.

[Sharma et al., 2007] has applied CBR in the domain of Real-Time Strategy (RTS) games. As described in section 1.1.1, strategy games are one of the game genres that has the largest state space. The real-time aspect adds even more complexity, as many possible actions can be done simultaneously. The authors objective was to build an artificial player with good performance level. CBR would avoid the need of an exhaustive exploration of the state space.

In this work, the context part of the cases is represented by a 5-dimension vector. Each of these dimensions represents a certain aspect of the game experience: measures on the player current state, or of its environment. Authors have shown increases in the performances of an artificial player on situations it never had seen before, by using knowledge from previous experiences.

[Ram et al., 2007] subsequently pointed out the difficulty of getting accurate initial cases. They proposed mechanisms to record experts playing the game, and to extract cases from it. The experts then only have to "show" what they want the game artificial intelligence to be, making the approach easy to implement on a given game (see table 2.6). The CBR allows to instantiate the experts provided strategies to situations in which the players are. Depending on the game, the amount of cases that need to be taught can be quite large. Moreover, differences between players are problematic, since they may require different game modifications in the very same cases. Moreover, it is likely that some aspects of the game cannot be controlled by the expert in a straightforward manner, specifically in the case where the adaptation mechanism does not involve solely opponents, but also environment variables which are not designed to be incarnated by intelligent players.

Table 2.6 — Case-Based Reasoning adequacy to the defined criteria for the game adaptation system

Characteristic	Adequacy	Comment
Genericity	$--$	Relies on the existence and importance of NPCs, requires demonstrable adaptation mechanisms
Ease of instantiation	$++$	Domains experts are allowed to demonstrate what they want
Deals with great complexity	$-$	Game states are classified. The task of experts grows with complexity
No strong assumptions on users	$++$	No user modeling
Handles unforeseen situations	$=$	Adaptivity if situations can be generalized

2.1.7 Conclusion on Learning

Approaches found in the literature that try to apply machine learning techniques to the problem of dynamic adaptation of games and serious games **focus heavily of artificial entities**. While there is sometimes encouraging results in these works, **a large amount of problems remains unaddressed**. In various kinds of simulation, the environment in which both the player and the artificial entities are situated has a significant impact on the difficulty of the task, as well as resources and other parameters that define the game experience. Therefore an approach that only sets the focus on autonomous artificial entities **cannot cover the adaptation requirements of such simulation games**.

Moreover, the nature of the machine learning field comes with constraints. The basic principle of the field is to **analyze data to gain generalization and prediction abilities**. In highly complex environments, **significant data may not be available**. If a search space is big enough, any learning is bound to fail. Another way to be able to work efficiently could be found in the ability to rapidly react to changes, rather than to anticipate them. For these reasons, other types of works are analyzed in the next sections.

2.2 User-Centered Computation

A large number of approaches have placed the human being in the center of their design. By emphasizing the need of taking the complexity and the diversity of the population playing a game into account, different means of adaptation can be designed.

2.2.1 User-Centered Game Design

[Magerko et al., 2008] have made a contribution in the field of educational games. They emphasize the fact that educational games are not necessarily a voluntary experience, and thus, may encounter a significantly diverse audience. Besides, there is a strong requirement that an educational game attains its objective of providing an improvement in the skills or knowledge of players.

These authors provide a survey of different studies of player learning styles. They highlight that different individuals may take advantage of different features of the same game, depending on their characteristics, preferences or personalities. They cite different studies, such as Dunn's learning styles [Dunn et al., 1981], Kolb's learning styles [Willcoxson and Prosser, 1996], and mention that other studies identify as many as 71 different learning preferences, potentially context-dependent.

In the context of their contribution, players are categorized into three profiles according to their motivation. Motivation can be *extrinsic*, such as the desire to obtain good grades or approval, or *intrinsic*, with the desire of doing well for its own sake. Among the extrinsically motivated players, they distinguish achievement-oriented players, that seek to complete various challenges, and winning-oriented players, that simply want to avoid loosing.

In order for a single game to provide an efficient experience, they propose this interesting idea that a game design should be an abstract object, that can be instantiated any

number of times, enabling different variations. The game they design is named SCRUB. It is a health-related mini-game that teaches how to effectively prevent the spread of pathogen organisms on our hands. In the design of the game, they identify a set of six features that can be activated or not. These features are related to both the *gameplay*, the *interface*, and the *knowledge* presented to the player.

From the domain knowledge and assumptions on how players think, each feature is analyzed to determine whether or not it is profitable to each player profile. Consequently, there are three instantiations of the game, one for each identified player profile.

The next step in the approach is to propose the right game instantiation to the right player. Two solutions are proposed. First, the choice can be given to the player. When the game starts, the player has to choose a profile from a list, and then plays the corresponding game. The second option involves a questionnaire, used to determine the category of the player.

As shown in table 2.7, this approach is quite rigid, since no modification in the selected feature occurs once the game is started. Rather than an adaptive game, this should be considered as an adaptable game. It offers means to be modified to better suit the player, but it does nothing by itself.

Interestingly, the idea of selecting features could be applied to a wide variety of games, even though it would be difficult to determine relevant player categories and related game features.

Table 2.7 — User-centered game design adequacy to the defined criteria for the game adaptation system

Characteristic	Adequacy	Comment
Genericity	++	Most of games have features that could be enabled
Ease of instantiation	=	No special skill is required to implement this approach. Questionnaires can be a burden
Deals with great complexity	−−	Dynamics of games are ignored
No strong assumptions on users	−−	Strong assumptions on players' desires and needs
Handles unforeseen situations	−−	No modifications once the game is started

2.2.2 Dynamic Modeling for Player-Centered Game Design

In the context of player-centered game design, [Charles et al., 2005] state that "every player is different; each has a different preference for the pace and style of gameplay within a game". They claim that closely related players, with similar abilities, may perceive a different challenge level on a given aspect of the game.

Moreover, they point out the difficulty to identify the desires and needs of human play-

ers. Questionnaires are not trivial, and they can be quite time consuming, and it can be unclear as if the questions are appropriate. They claim that the generally used player types tend to be overly simplistic, as they cannot anticipate all different styles, nor all the factors involved in the characterization of the gameplay experience.

For these reasons, they advocate the usage of dynamic modeling of users. With dynamic modeling, the consideration of a player, such as a profile, or other characteristics, may evolve while the game is played. This would allow the model to more accurately deal with the dynamical aspects and variety of the players population.

Charles et al. suggest that any game adaptation mechanism could benefit from dynamic modeling. While they do not actually propose adaptation mechanisms, they describe a "potential framework for an adaptive game system", in which the result of game adaptation is used to update the model of the player. A player could be assigned to another category or profile, or a new category/profile could be created if none fits the current player. Consequently, there is a requirement for real-time adaptation, rather than to try to design an adapted challenge before the game is actually played.

To design player profiles, they propose to list characteristics, such as combat abilities or puzzle solving skills, and to correlate them with observable in-game variables. A mapping should be designed between game observables and characteristic values. Then, the game should be designed with regards to the profile.

As they point out, these characteristics can be highly correlated with a specific game genre. It is unclear how a designer is supposed to choose relevant high-level characteristics, given the difficulty of the problem as we described it before. Moreover, selecting relevant variables, and designing a relevant mapping between them and the characteristics is far from trivial. Finally, how the profiles should be used to modify the game experience is also unclear.

If the idea of dynamic modeling is interesting, too few elements are available to start implementing it, even in a simple game. Therefore we are not able to provide an analysis through our set of criteria.

2.2.3 Scenario Rewriting

Several approaches have tried to dynamically modify the game experience by modifying the sequence of events the player are confronted with. The term "scenario" can be used to describe a story or a timeline, in which the player has to interact with the game world by using various game mechanisms. The timeline concept can be applied in a large variety of games. A way to adapt the game experience to the skills and abilities of a player is thus to modify this timeline according to the actions and the characteristics of this player.

Niehaus et al., in the context of learning games, have proposed a formalization of scenarios as sequences of plans [Li and Riedl, 2011; Niehaus and Riedl, 2009]. A plan is composed primarily of an event that is supposed to happen in the game, and which necessitates actions from the player. The concept is quite general and can find an equivalent in many different genres. It can be a fight with an enemy, it can be a specific manipulation of the environment, or any new goal presented to the player. An event generally has constraints, such as precon-

ditions in the game environment or in the player's achievements, or temporal constraints defining orders between events to preserve the consistency of the scenario.

From the domain knowledge, various skills that the learning game intends to teach to players are identified. Each skill is connected to one or several events. Therefore, a given scenario, which is a sequence of events, can also be considered as a sequence of needed skills to complete the scenario.

The main idea for adaptation is that different players require different scenarios. Depending on the abilities of the player, different skills should be practiced. Consequently, Niehaus et al. propose two main operations on the scenario: addition and deletion. Given a set of pre-authored events, associated with the skills they involve, when a player needs to practice a specific skill, it is possible to add an event in the scenario involving this skill. Similarly, when a player is already proficient at a certain task, and no longer needs to practice a particular skill, events in the scenario that are focused on this specific skill can be removed.

Thanks to the definition of events, with temporal and causal constraints, the consistency of the scenario can be preserved while performing addition and deletion operations. These operations are effectively executed only if all constraints on events can still be satisfied.

Automation of scenario rewriting can lead to a large number of scenarios, depending on the number of pre-authored events, and as shown in table 2.8, complexity can be dealt with since each scenario fragment is considered separately. However, if the task selection is efficient to work out relevant skills, there is no difficulty adaptation within a given task. This approach may lead to the case where a player is presented a task that implies practice for actually needed skills, but the challenge level of the task itself is not adequate.

Table 2.8 — Scenario rewriting adequacy to the defined criteria for the game adaptation system

Characteristic	Adequacy	Comment
Genericity	$--$	Sole focus on scenario fragments reordering
Ease of use	$++$	Scenario description should be familiar to game designers
Deals with great complexity	$++$	Scenario fragments are considered individually
No strong assumptions on users	$+$	Only acquired skills are listed
Handles unforeseen situations	$--$	Dead-end can occur in scenario if unadapted fragment

2.2.4 Conclusion on User-Centered Computation

When user's satisfaction is placed at the center of the design process, this can lead to satisfactory results. The design of the game itself revolves around having a proper challenge for the player to face. Many approaches that fall in this category **make an extensive use of certain aspects of the game that are relevant only to their specific application**. In other words, these are often *ad hoc* approaches, with a **low level of reusability**. Consequently,

even if some of these works successfully address the challenges they were interested in, there is still room for improvement. The need for some more general adaptation mechanisms remains.

Moreover, and as we stated before, in the context of complex games with many interconnected aspects, it may be a good idea not to **make assumptions on how players think, act or behave under any circumstances**. If game adaptation mechanisms are to be deployed on large-scale video games, there may always be individuals that do not match in pre-established patterns.

2.3 Control and Optimization

Given a more abstract definition of our problem, it is possible to study different approaches. By considering the game and the player as a system, with inputs, outputs, and requirements on both, we can not only gain a high level of genericity, but also benefit from the work of control engineering and optimization, which are well investigated fields.

The nature of games implies that it is not always possible to apply all approaches from control and optimization – as we will see in the next sections – but several aspects of some of the most recent works can be used to adapt game experience to players.

In this section, we will review the major works of control engineering and optimization, and review approaches that have been applied in the context of games. Approaches coming from the classical control theory and control engineering are first described, as they are widely used in the industry. Techniques coming from Artificial Intelligence such as expert systems, evolutionary algorithms or neural networks are then studied, since they can show interesting advantages when dealing with nonlinear systems. Finally, a representative example of trajectory-based optimization techniques is detailed.

2.3.1 Classical Control Theory and Engineering

From the most general point of view, control theory is a discipline that seeks to control dynamical systems, generally referred to as *plants*. In this discipline, a target system is supposed to have an input, on which a control can be applied, and an output which can be measured. An objective – here named a *reference* – is usually set on the output of the target system. As opposed to the *open loop control* – where a predetermined control is applied with no consideration of what actually happens in the system – the emphasis in control engineering is set on *feedback loop control*: the control seeks to have the actual output of the target system match the given reference.

2.3.1.1 General Description

The general idea of techniques coming from the control engineering is to build a controller that takes a measure of a target system, and compares it with the reference value. From these two values, an error can be computed. From this error, a proper control is applied on an input of the target system (see figure 2.5).

The most famous approach following these principles is the *PID Controllers* method [Astrom and Hagglund, 1995]. *PID* stands for *Proportional-Integral-Derivative*. The main idea is to transform the error measured on a target system output (i.e. the difference between the actual value and the reference) with a sum of three functions that depend on the time t. $e(t)$ is the error at time t, and the three functions are defined as:

- $P(t) = K_p e(t)$ is a value proportional to the error.
- $I(t) = K_i e(t)$ is the integral of the error, which enables the consideration of the duration of the error.
- $D(t) = K_d d/dte(t)$ uses the derivative of the error, enabling the consideration of its variation over time.

A combination of these three functions gives a correction $M(t)$ to apply on a numerical input of the target system:

$M(t) = P(t) + I(t) + D(t)$

The control applied on the input is therefore determined by the values of the three constants K_p, K_i and K_d. Finding a proper policy on such systems consist in finding appropriate values for the three constants composing the correction function. Several methods exist to find appropriate values for the constants, from manual tuning, to computer simulation or with more formal methods [Tyreus and Luyben, 1992].

PID controllers are often used on systems in which control needs to be applied on one numerical input given the value on one numerical output (sometimes referred as SISO: Single-Input and Single-Output). A good knowledge on the system to control can be a precious help to find optimal values for the constants of the controller.

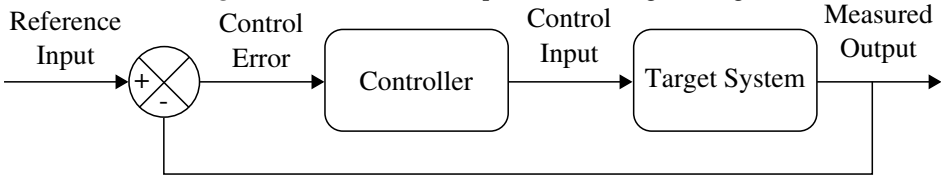


Figure 2.5 — Feedback Loop in Control Engineering

2.3.1.2 Analysis

PID controllers offer a real level of genericity, since they can be applied on various kinds of systems. Their privileged domain is the mechanical domain, but there is no obstacle for them to be applied to any other field, since they do not necessarily require concepts of the domain they are applied to, and they only manipulate numerical values.

Since PID controllers can only manipulate numerical values, some work has to be done to be able to apply them on games. We have mentioned systems that can be in several states, and expressed a requirement for a control over these systems that would find an optimal state sequence. Therefore, there is a need for a mapping between the concepts manipulated in the target system and the logic of PID controllers.

However, considering the type of systems we want to control, there may be other difficulties in the application of PID controllers. The systems we have described typically offer many control points and exhibit a large number of measurement points. Multi-Inputs and Multi-Outputs (MIMO) systems constitute a limitation of PID controllers.

For instance, combining PID controllers in possible, potentially in cascade, except that the tuning of these controllers becomes problematic. In practice, constants of PID controllers can be found by experience, with a knowledge of the target system. In our case, there is no knowledge of the target system, and it is one of our requirement not to make too many assumptions about it.

Other methods, like Ziegler-Nichols method [Ziegler and Nichols, 1993], allow to algorithmically find appropriate values. However, these methods may take a significant amount of time. In this case, when trying to control a system like the one we have described, the real-time requirement would make these approaches unpractical. The optimal values for the constant of a PID controller may change over time, according to the internal state of the system, and therefore, the algorithm would have to be run continuously to update them, which is unrealistic given the time it takes.

Overall, it can be stated that the complexity, and more importantly, the dynamical aspects of the games in interaction with human players tend to make it quite difficult for the PID controllers to be used to efficiently control them (see table 2.9).

Table 2.9 — Control engineering classical approaches adequacy to the defined criteria for the game adaptation system

Characteristic	Adequacy	Comment
Genericity	++	Does not depend on the domain
Ease of instantiation	−−	Needs a mapping between game concepts and PID concepts
Deals with great complexity	−−	Designed for single input/single output linear systems
No strong assumptions on users	+	Can work on “black-box” models
Handles unforeseen situations	−−	Cannot deal with change of dynamics

Given the difficulties appearing when trying to control systems with a high level of complexity, the control field has evolved towards a greater use of techniques coming from the artificial intelligence field, leading to what is now referred to as Intelligent Control (IC). In IC, there is no need for an expert to provide a comprehensive model of the target system. Rather than that, the technique used builds its own abstract model of the target system. This has been referred to as an intelligence shift, where the intelligence does not come from the designer, but from the technique. IC techniques can be inspired by the way human beings reason about problems they try to solve, or by how biological systems deal with the control problem [Smith, 2002].

The next sections describe representative approaches of intelligent control: expert systems, evolutionary computing and artificial neural networks.

2.3.2 Expert Systems

Some artificial or natural systems are not truly understood, but are still controlled by humans experts, such as electrical power plants, chemical refineries or air-traffic control. These experts have some kind of intuitive understanding of how these target systems work, but they are not able to accurately express how they work, they are not capable of providing a model of these systems [Dvorak, 1987]. Experts systems try to take advantage of this knowledge to efficiently control various kinds of system.

2.3.2.1 Principle and Architecture

Expert systems try to make use of empirical or intuitive knowledge for the control of systems that cannot be modeled. Expert systems generally consist of two main parts: the rule base and the inference engine.

The rule base represents the knowledge that one or several experts can express about how to control the target system. This knowledge does not represent the inner functioning of the target system, but it states how experts would deal with it, in a language relatively similar to the natural language. Rules generally take the form of logical rules:

$$IF\ condition\ THEN\ action$$

Conditions can be expressed with propositional logic, first order logic, high order logic or others, depending on the domain of the target system and on the nature of the knowledge of the experts. Experts express their knowledge by writing all the appropriate rules.

The purpose of the inference engine is to use the data given in input by users, and to use the rule base to produce new knowledge. An inference engine may be complex to develop, depending on the type of rules it has to manipulate, but it does not depend on the specific rule base it has to work with. Therefore, once an inference engine is developed, it can be used in many different expert systems.

In order to control an external system, the expert system needs to be connected to both a user and the inputs and outputs of the target system (see figure 2.6).

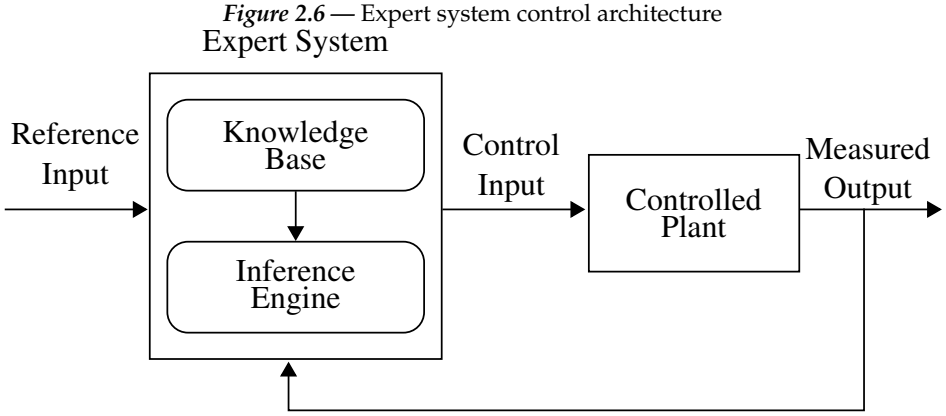
The user of an expert system base sets the desired reference for the outputs of the target system. According to the state of the target system and the references, the knowledge base is used by the inference engine to determine what are the appropriate actions to influence the target system towards the reference.

2.3.2.2 Fuzzy Logic

It could be argued that the use of logical rules in the knowledge base of expert systems is not always easily compatible with the way humans work, as they sometimes reason in a less formal way. To reduce the gap between human reasoning and the rule base formalism, fuzzy logic has been advocated as one potential solution.

The term fuzzy logic has been introduced by [Zadeh, 1965], in the theory of fuzzy sets. A fuzzy set is a set whose elements have degree of membership. More formally, a fuzzy set

Figure 2.6 — Expert system control architecture

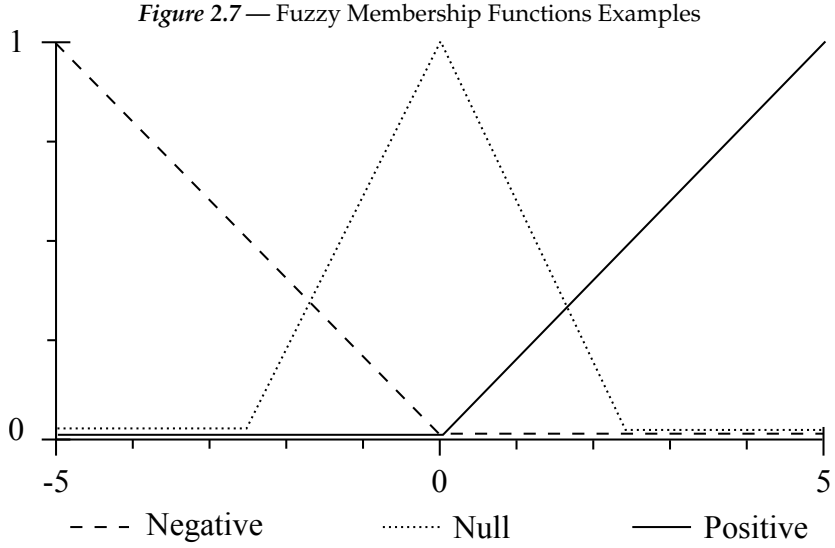


has a definition domain, and a membership function which, for each element of the domain, yields a value between 0 and 1 named the *grade* of membership. An element that has a grade membership of 0 is not included in the fuzzy set. If its grade is 1, then it is fully included in the set, and in all other cases, the element is said to be a fuzzy member of the set.

Similarly, fuzzy logic does not only consider the *true* and *false* elements, but instead that an element can take an infinite amount of intermediate values. Membership functions are used to map elements of a definition domain to a value between 0 (false) and 1 (true). An interesting aspect with this logic is the possibility for an element to have several properties that would generally be exclusive.

The example in figure 2.7 shows a definition domain which is a subset of real numbers: $[-5, 5]$. The dashed curve defines the fuzzy set (or the membership function of the set) named "negative", the dotted line defines the fuzzy set named "positive", and the thick line defines the set "null". Therefore, each element in the domain $[-5, 5]$ has a membership grade for the sets *negative*, *positive* and *null*. Depending on the domain, these fuzzy sets can be useful, as they introduce flexibility in the manipulated concepts.

Figure 2.7 — Fuzzy Membership Functions Examples



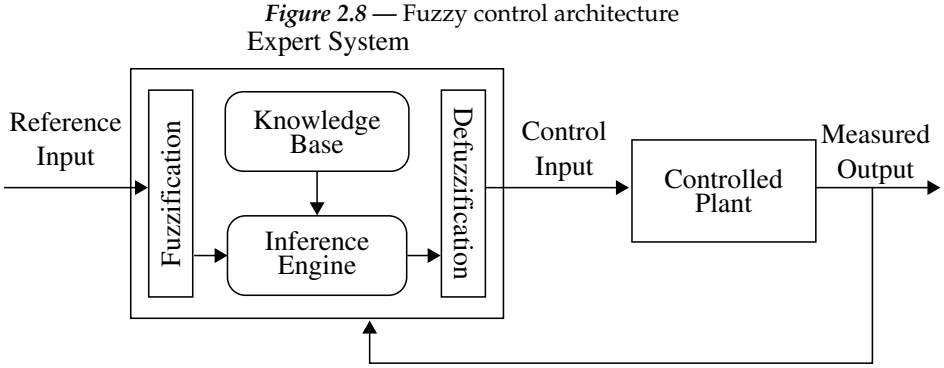
When designing an expert system, writing the rule base can be made easier with the in-

troduction of fuzzy logic. Experts of the domain may not want to express extremely accurate concepts, but instead more continuous properties.

In order for an expert system using fuzzy logic rules to be applied in the control of a target system, there is a need for conversion between the target system and the fuzzy rules. This step is referred to as *fuzzification* (see figure 2.8). It is a mechanism which takes values of the target system, and uses membership functions to determine the grade of each predicate of the knowledge base.

The inference engine takes the fuzzified knowledge and deduces applicable control for the target system. The actual control to apply on the target system is deduced from the different grades of the conclusions of the inference engine. Several techniques exist to get from a set of graded conclusion to an actual numerical control. This process is referred to as the *defuzzification*.

Figure 2.8 — Fuzzy control architecture



2.3.2.3 Analysis

If we are not currently aware of an application of expert systems on the control of video games, they still seem an interesting approach for whom is interested in the control of games or simulations. As shown in table 2.10, expert systems are quite generic, since they express the knowledge of the experts with a set of logical rules. They can easily be tuned or modified, as experts manipulate knowledge in a flexible way. However, if no model of the target system is required, control abilities should be available before using an expert system. If experts are not able to control the system manually, then no expert system could be built to replace them. It is a good solution to make expert control policy automatic and to scale it up, but it does not go further than human skills and expertise.

Additionally, if writing a proper rule base and tuning it to be able to efficiently control a target system is a feasible thing, it still requires a significant amount of work, making it impossible to do in real-time. Therefore, the expert system control lacks adaptivity. If the dynamics of the target system changes over time, the rule base is eventually going to be outdated, making the control system inefficient.

The fuzzy logic extension offers additional advantages to the expert system approach, and even though they have been largely controverted [Zadeh, 2008], fuzzy logic controllers encountered success in many domains, and they are still widely used today in the indus-

try. Expression of knowledge using membership functions is a significant advantage of this approach, and it facilitates considerably the work of experts.

However, fuzzy logic does not overcome the general problem of expert systems, as it does not prevent the approach from needing a human knowledge on how to efficiently control the target system, nor it gives it the possibility to deal with evolution of target systems.

Table 2.10 — Expert systems and fuzzy controller adequacy to the defined criteria for the game adaptation system

Characteristic	Adequacy	Comment
Genericity	++	Does not depend on the domain
Ease of instantiation	++	Concepts are close to domain concepts
Deals with great complexity	−−	Need for a human expertise and exhaustiveness
No strong assumptions on users	++	No modeling of the target system
Handles unforeseen situations	−−	Cannot adapt to dynamic modification

2.3.3 Evolutionary Computing

Other techniques in artificial intelligence applied to control take their inspiration from natural phenomena. It is however important to keep in mind that this is just a loose inspiration and not a simulation of biological systems. The first mentioned approaches are inspired by genetics and species evolution, while the others take their inspiration in neural systems found in brains.

2.3.3.1 Genetic Algorithms

Genetic Algorithms (GAs) are inspired by evolutions of species and genetics. The *survival of the fittest principle* is used to produce satisfactory solutions [Holland, 1975].

A GA consists of a population of solutions that evolves from randomly generated individuals to a set of supposedly satisfactory solutions for a given problem. The solutions are not directly designed, but instead, an evolutionary process is developed to enable new solutions to be created from old ones as they are evaluated by an external process.

The first step in the design of a GA is the definition of how the potential solutions to the problem are represented. As a reference to genetics, the representation of a solution is referred to as the *genotype* and the actual corresponding solution as the *phenotype*.

To be usable, a GA also needs a fitness function, or an evaluation function that is able to state how good a solution is performing when trying to solve the problem.

Finally, means of evolution are required to create new individuals from previous solutions. This is done thanks to genetic operators, namely, crossover and mutation. Crossover takes a couple of genotypes, and merge their characteristics to obtain a new solution, with hopefully the advantages of both the original solutions. A mutation operator changes small parts of the genotype of a single individual. Its purpose is to preserve diversity within the

population.

When these operators are defined, the GA can be executed. First, a random population of solutions is generated. Then, all the individuals are assigned to a fitness value using the evaluation function. Generally, the weakest part of the population is eliminated. From the better part of the population, and with the help of genetic operators, a new set of individuals is generated, and reinserted into the population so as the size of the population remains unchanged.

These steps represent one iteration of the algorithm. Usually many iterations are needed before reaching a satisfactory solution.

A termination criterion is used to determine when enough iterations are done, which could be a threshold for the fitness value, or a maximum number of iterations. The best solution is finally selected in order to deal with the initial problem.

GAs have proven to be able to control a large variety of target systems [Fleming and Purshouse, 2001]. Provided that a control policy can be represented with any kind of genotype, then it can be manipulated by a GA. The advantage is that a solution does not necessarily need to be of a numerical form, but can be represented by more domain-related concepts. Moreover, GAs can be used to control systems with a significant degree of nonlinearities, noise or randomness, and of which no model is available to predict the outputs of the system. Thanks to the evaluation of solutions, GAs will still be able to outperform exhaustive searches or similar methods.

The specification of the fitness function confers another advantage to GAs. Indeed, this function states how well a solution is, therefore it is more straightforward for the experts to transfer their knowledge into the control system. Evaluation criteria can be defined on several aspects of the solutions, giving more flexibility to experts than traditional control approaches.

[Zook et al., 2012] have used GAs in the context of games to generate appropriate content, adapted to a player's skills and abilities. Their work takes place in the context of strategy games, in which several entities evolve in a common environment, each of them having various goals. A scenario is considered to be a parametrization of a specific mission. Adapting the difficulty of the game to maintain the user in satisfying conditions then consists in finding the proper parameters values of the mission.

These authors propose that game designers specify a performance curve that the player should experience. An interesting concept is that they talk about performance, and not about difficulty. Indeed, difficulty is largely subjective, and what is considered difficult for a given individual may not by another individual. Performance on the other hand, is a measurable value, and reflects the actions of a player on the game as it was experienced. Consequently, when a designer expresses a desired performance, all players should experience it, and therefore have the same measurable results regardless of their skills and abilities.

A GA generates satisfying scenarios after several hundreds of iterations. At each iteration, the quality of the scenario has to be evaluated to proceed to the next iteration. For that reason, they used a player model in order to predict how well the player is going to perform on specific tasks. These prediction abilities prevent the player from having to play the game

as many times as needed by the GA, which would be not only impractical, but also bound to fail, as the player evolves while playing.

It could be argued that the design of the representation of the solution, i.e. the phenotype, already consists of a first step of the resolution process. By determining the characteristics to be coded on the string-based representation, needed characteristics of the solution are indirectly specified. For instance the solution complexity is fixed. To overcome this drawback, a GA enhancement has been proposed: Genetic Programming.

2.3.3.2 Genetic Programming

Genetic Programming (GP) is a specialization of Genetic Algorithms. Basic principles are the same, a population of solutions is evaluated with a fitness function and then used to create new individuals. However in GP there is no fixed representation of solutions. Instead the evolving genotype structure is a tree, of which the nodes are operators. Operators can be simple arithmetical operators or more complex operators inspired by domain knowledge.

The initial population of solutions is composed of simple random solutions, and as the number of generations rises, solutions are combined with each other to create more and more complex, and hopefully, more satisfactory solutions to the problem.

The advantage of GP over GAs is that no constraint on the nature of the solution arises from the choice of the solution genotype. Solutions can take more versatile forms, in addition to reach arbitrary levels of complexity. As [Koza, 1992] said "the size, shape, and structural complexity should be part of the answer produced by a problem solving technique – not part of the question."

There is still an interesting interrogation about how a solution can effectively be associated with a fitness value if no model of the target system is available. If no analytical method is able to predict the quality of a solution, then the solution needs to be actually tested on the target system. This can lead to effective solutions, but one should bear in mind that many iterations will be needed by the algorithm before it finds a satisfying solution. Depending on the target system, it may not be possible to evaluate solutions as many times as required by the algorithm.

2.3.3.3 Analysis

As depicted in table 2.11, the evolutionary computing paradigm has several advantages over other approaches. It can be used to address problems with no known solutions. For instance in the context of control, the sole definition of a fitness function can lead an evolutionary algorithm to find a satisfying solution, that is a satisfying control policy.

Within this paradigm, high complexity does not prevent a solution to be found. With genetic programming, the complexity of the solution can be increased to an arbitrary degree in order to provide a satisfactory answer to the problem.

Additionally, evolutionary algorithms are domain-friendly. They can handle highly heterogeneous constraints and objectives, as they do not need an analytical expression of the domain. As long as a solution can be represented in some way, then it can be processed by

an evolutionary algorithm.

However, there is a major drawback. Before any satisfactory solution can be found, there must be a large number of iterations, each one of them consisting in creating a solution and evaluating it. In the case of a problem involving human individuals, evaluating a large number of solutions could only be acceptable if the problem involved was static. In the case of a dynamical problem, with potential sudden changes in the needed solution, a large number of evaluations cannot be acceptable. Some have tried to compensate this aspect of the problem by building predictive abilities on the players' evolutions, but again, this is contrary to the requirements we have expressed in 1.4, as we claim that they would be impractical for a game with a large number of dimensions, and a population with a significant diversity.

Table 2.11 — Evolutionary computing adequacy to the defined criteria for the game adaptation system

Characteristic	Adequacy	Comment
Genericity	++	Any aspect of gameplay could be dealt with
Ease of instantiation	++	Concepts from the domains are manipulated in both solution expression and fitness
Deals with great complexity	++	Arbitrarily complex solutions can be found
No strong assumptions on users	+	*A priori* no need for information about players
Handles unforeseen situations	−−	Need for enormous amount of time and data

2.3.4 Artificial Neural Networks

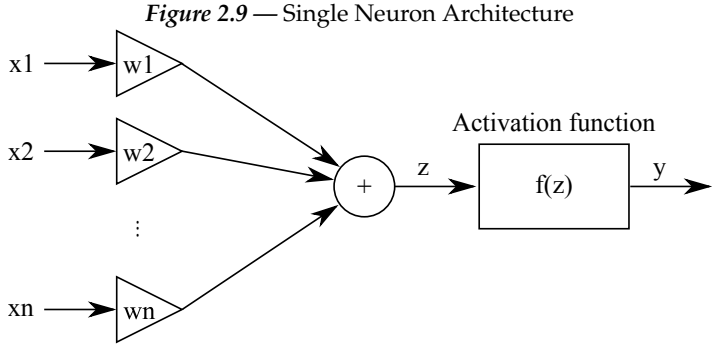
On a different time-scale, interconnected biological neurons forming humans and animals brains have inspired computational models. Artificial Neural Networks (ANNs) try to reproduce the organization of real neural networks to gain control or prediction abilities of complex functions. They have proven useful in numerous domain, such as pattern recognition, function approximation, and they are also used in control problems [Dayhoff and DeLeo, 2001].

2.3.4.1 General Description

As seen in figure 2.9, a single neuron can be seen as a binary function with one output (y) and one or several inputs $(x1, ..., xn)$. A weight $(w1, ..., wn)$ is associated with each of the inputs of the neuron. The weighted sum of the inputs goes through the neuron *activation function* that determines the output sent by the neuron.

Structurally, a neural network is composed of an interconnected set of neurons. Three types of neurons are usually distinguished, sometimes organized in separate layers. The

Figure 2.9 — Single Neuron Architecture



first type of neuron is referred to as the *input layer*. These neurons are directly connected to the environment, as they are the first to receive the signal given to the network.

Similarly, the *output layer* is defined as the set of neurons that ultimately sends a signal outside the network.

In between these two layers, there can be any number of interconnected neurons. They can be organized in layers, in which case they are referred to as *hidden layers*.

A signal that goes through the input layer of the neural network triggers activity and can potentially generate an output signal if an output layer is present in the structure. Consequently, the ANN is seen as a system, whose functionality can be adjusted by modifying either the weights of the neurons, or the topology of the network.

The difficulty when using artificial neural networks consists in setting the adequate values for the weights of the network so its functionality becomes useful for the intended purpose. Different learning strategies exist. For that purpose there are different strategies that can learn weights. The learning process is often referred to as the *learning phase*, as opposed to when the network is actually used, which is referred to as the *exploitation phase*.

Unsupervised learning is a set of techniques based only on the inputs given to a neural network. In unsupervised learning, no feedback from designers or from the network environment is used. These techniques are not given any information regarding the quality of the solution, nor the direction in which it should be modified.

A common unsupervised learning technique is the *Hebb's rule*, or *Hebbian Learning*, which is often summarized as "Neurons that fire together, wire together. Neurons that fall out of sync fail to link". In other words, when a neuron is repeatedly involved in the activation of another neuron, the weight of the connections between these two neurons is strengthened.

Hebbian learning is used in a variety of neural networks, with different purposes. A good example is the Hopfield network. The Hopfield network is a fully connected neural network, where a neuron has only two possible states, either activated or not. The network is used as a content addressable memory: in the learning phase, several inputs that need to be *remembered* are given as inputs to the network. Since the network is fully connected, this generates activity for several time steps. The network finally gets stabilized in a certain state, one for each of the given inputs.

In the exploitation phase, when an incomplete or noised input is given to the network, the generated activity will make the network evolve, until it finally reaches a stable state.

That stable state is then associated with an input learned in the training phase. This is notably used in pattern recognition [Carpenter, 1989].

Another example of unsupervised learning is the Kohonen maps, also referred to as *Self-Organizing Maps* (SOM) [Kohonen, 1982]. SOMs consist of two layers: an input layer of n neurons and a hidden layer of m neurons. All the neurons of the input layer are connected to all neurons of the hidden layer. In the hidden layer, the notion of topology is added: neurons are placed in a (usually bi-dimensional) space, and the notion of distance between neurons is defined. Each neuron of the hidden layers has a randomly initialized weight vector of dimension n. When an input vector comes in the network through the input layer, the neuron with the weight the most similar to the input is selected. Once selected, the weight of the selected neuron as well as those of its neighbors are slightly modified toward the values coming from the input layer. The closer to the selected neuron a neighbor is, the most its weight is adjusted. A SOM is a way to map values from a n-dimension space to values of a m-dimension space. It is often used to visualize high-dimensional data in two dimensions.

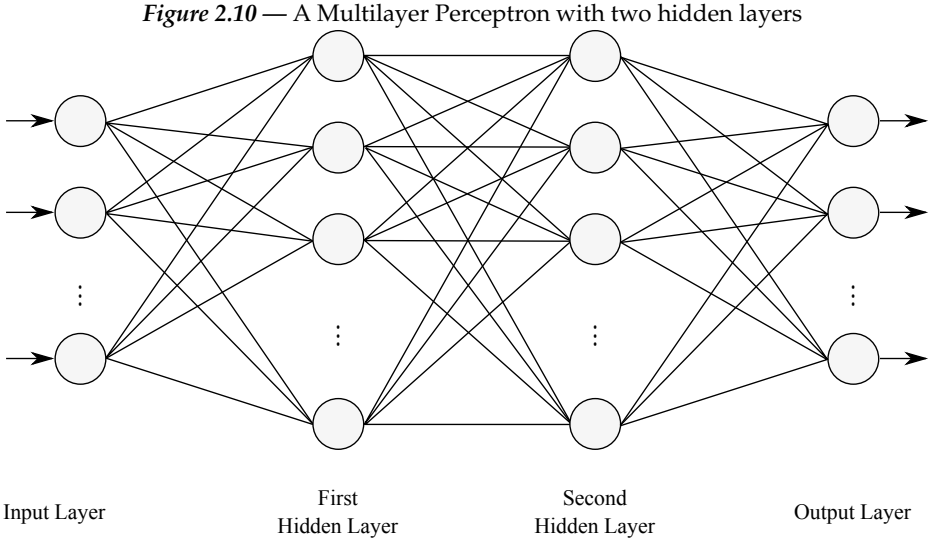
Supervised learning techniques train the network using external information. Basically, the training consists in confronting the ANN to data, and in using an algorithm to modify the weights of the network so that its output matches the data. If the training is properly achieved, the neural network gains the ability to reproduce the function it has been shown through the data, i.e. it is able to yield the same output for the same input, but more importantly, the network is able to generalize the data it has been confronted to.

The simplest model of ANN for supervised learning is the perceptron. Introduced by [Rosenblatt, 1958], the perceptron is composed of a single neuron with n inputs. A weight wi is associated with each input. The weighted sum of the inputs goes through the activation function. The perceptron is confronted to a training set of data composed of pairs of inputs and desired outputs. For each input, the neuron actual output is computed, and compared to the desired output so an error can be calculated. The weights of all inputs are adjusted according to the error. The bigger the error, the bigger the adjustment on weights.

To deal with more complex problems, the perceptron is extended to what is generally referred to as the Multilayer Perceptron (MLP). The MLP consists of separate layers, where connections between neurons do not form cycles (see figure 2.10). It has an input layer, from which the signal enters the network, an output layer from which it goes out, and one or several hidden layers. Similarly to the single layer perceptron, when an error is computed on the output of the network, the weights of the neurons in the output layers are modified according to steepness of the activation function. From the output layer, the error is backpropagated to previous layers to adjust weights of all neurons in the network.

The number of layers in the MLP and the training data needed for an appropriate training can be significantly high, however Cybenko's theorem states that any arbitrary function can be approximated using a neural network [Cybenko, 1989].

Figure 2.10 — A Multilayer Perceptron with two hidden layers



2.3.4.2 Control with Artificial Neural Networks

Artificial Neural Networks have been used in the context of control, generally with supervised learning techniques, by creating an inverse model of the target system. An inverted model of the target system allows to find the proper input to give to a target system in order to obtain the desired output [Hagan and Demuth, 1999].

The inverted model of an arbitrary system can be obtained by training a neural network with the actual system in two ways. Figure 2.11 shows the *General Training Architecture* where the neural network is given the output of the target system, and tries to predict the corresponding input. The difference between the predicted input and the actual input, i.e. the error, is used to train the neural network.

Figure 2.12 shows the *specialized training architecture*, where what is wanted from the target system (the reference) is given to the neural network inverse model. The ANN then yields what it assumes to be the corresponding input, and sends its to the target system. The target system returns its actual output, which is used in comparison of what was expected to compute an error. The error is then passed to the neural network to train it.

Figure 2.11 — General Training Architecture

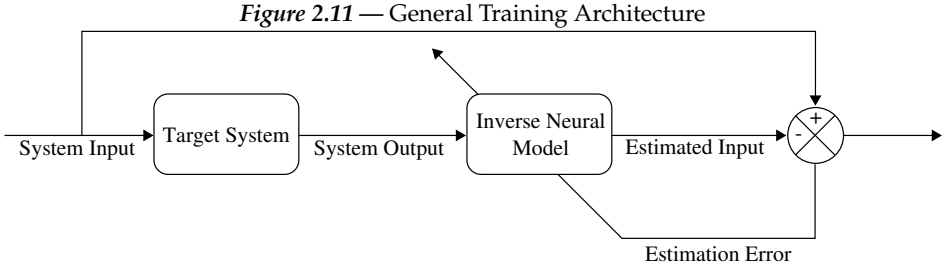
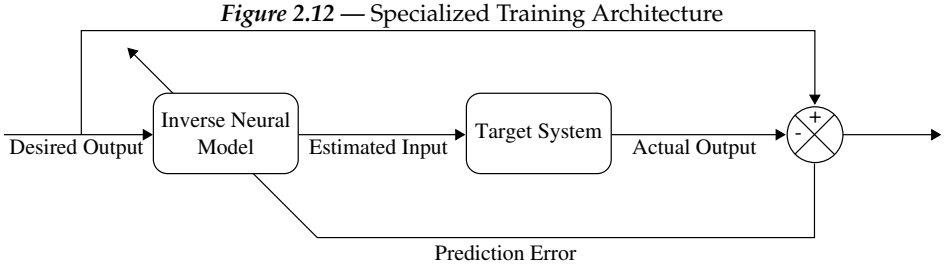


Figure 2.12 — Specialized Training Architecture



2.3.4.3 Applications on Video Game Adaptation

An extensive usage of Artificial Neural Networks has been done by [Stanley et al., 2005] to design an adaptive game experience. This work takes place in the context of the NERO (NeuroEvolving Robotic Operatives) game. The main idea is to design a game in which several agents must be trained by the player. Each of these agents features its own neural network, that will receive the feedback of the player as its learning input.

The particularity of these neural networks is the way they are learning. Rather than just learning new weights, there is also a change in the topology of the network. It starts with a simple topology (only a few neurons organized in two layers), and when the limit of what the NN can produce is reached, the topology complexity is increased.

Evolutionary algorithms are used to determine the best topology of the network. Each neural network is exhaustively described by its genes. These genes consist of a textual description of all the nodes present in the ANN, along with the description of their connections between each other, as well as the weights of all these connections. An evolutionary algorithm combines and mutates genes, to generate new neural networks with different topologies.

A contribution made by this work is the use of the evolutionary algorithm in parallel of the utilization of the produced solutions. That is, instead of running the algorithm for a fixed time and then using the best solution found, the algorithm will be used at runtime. Every n ticks of the system clock, all solutions are evaluated, and the worst solution is removed and replaced by a newly generated individual. The notion of lifetime is introduced in order not to remove recent solutions that yet have to be improved. This mechanism intends to give solutions enough time to learn from the feedback of the user.

Artificial entities populating the game environment start with no skills, and only the ability to learn. The player designs the situations in which these entities are, by placing various game artifacts in their environment, and let them learn and develop new skills. Authors have shown that complex behaviors such as path-finding abilities or enemy tracking could be discovered in a few minutes.

However, it seems that the adaptivity abilities of this work are quite limited to a specific area of games, in which players need to actively train artificial agents. It is yet to be demonstrated if other types of gameplay could benefit from this work, and how to adapt artificial entities to players, in order to provide adapted challenges.

Besides, even if the learning time is relatively short, care must be taken about what is the behavior while the system is still learning. Random behaviors may appear and, depending

on the game and situations, they could have unwanted consequences. One must evaluate the feasibility of such an approach in each particular situation.

2.3.4.4 Analysis

One major issue when dealing with supervised learning in neural networks is *overlearning*. Overlearning occurs when noise is present in the training data. It is possible that too many training iterations occur, resulting in the network learning and reproducing noise. When an ANN has overlearned, the network keeps getting better on predicting the learning data, but the performance on data outside the training set is decreased. Optimally training a neural network usually requires skills and experience.

Table 2.12 shows that similarly to evolutionary algorithms, inverted neural networks can help in the control of systems for which no model can be established, even if no experts are capable of specifying an efficient control policy. However, these methods require a large amount of training data, which may not always be available, especially when controlling systems in interaction with human beings. Additionally, a significant amount of time is needed for a neural network to be appropriately tuned. Therefore if the target is changing over time, what has been learned in the training phase may not remain adequate, making it difficult for an artificial neural network to be used to control dynamical systems.

Table 2.12 — Artificial neural network adequacy to the defined criteria for the game adaptation system

Characteristic	Adequacy	Comment
Genericity	++	Does not depend on the domain
Ease of instantiation	−−	Needs expertise to instantiate and to train the network
Deals with great complexity	++	Virtually any solution can be obtained
No strong assumptions on users	++	No understanding of the target system is needed
Handles unforeseen situations	−−	Needs significant amount of time and data

2.3.5 Trajectory Based Optimization Methods – Example of the Gradient Descent Algorithm

Since the beginning of this section on optimization, we have only mentioned *population-based* methods that share many characteristics. More specifically, they all require the evaluation of many different solutions before being able to decide what is the most optimal choice. When the evaluation of the quality of a solution is non trivial, like when a human is involved, it can be a better idea to pick one single solution, and to improve it over time. If the improvement can be done fast enough, then this approach could be a satisfying option.

2.3.5.1 General Description

Trajectory-based approaches consist more in providing adaptive solutions rather than learning good solutions. In this section, we introduce one famous trajectory-based optimization technique: the gradient descent algorithm. Even though the gradient descent – sometimes referred to as the *method of the steepest descent* – is a specific algorithm, it is worth being described in this section, as it is representative of a broad area of continuous optimization, and it is widely used in many different fields, in addition to video game adaptation.

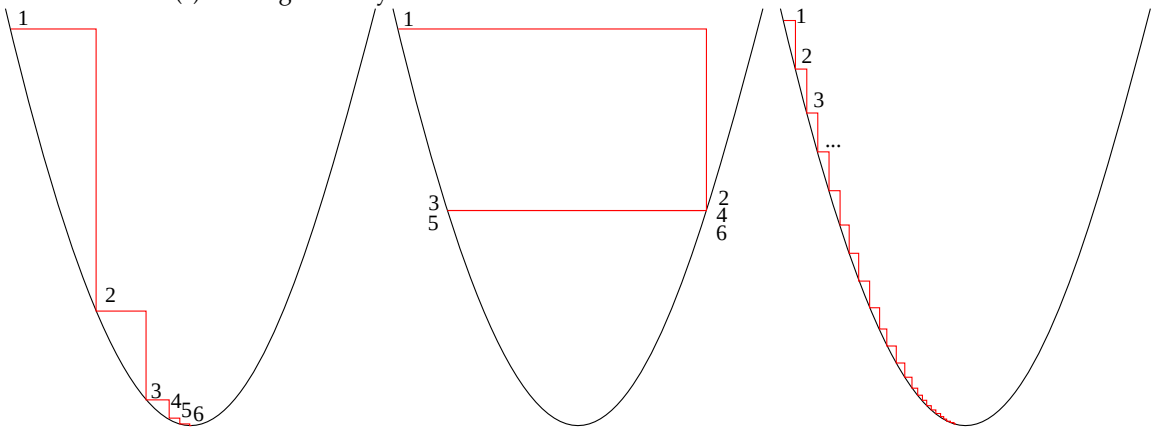
The goal of the algorithm is to find the minimum of a given continuous derivable function. Finding this minimum analytically can be quite challenging, if not impossible. The gradient descent method seeks to find the minimum procedurally.

Considering a function $f : E \to F$, the algorithm will pick a random value $x_0 \in E$, and compute the gradient $\nabla f(x)$ (the derivative). Each value is calculated based on the previous one, so that $x_{i+1} = x_i - \eta \nabla f(x_i)$.

The algorithm is stopped either if i is higher to a certain threshold, or if the gain of each iteration is lower than a certain other threshold. η is a parameter that determines the convergence speed. The bigger η is, the faster the change of the value. Figure 2.13a shows the trajectory of the algorithm with a relatively large η value, whereas, figure 2.13c shows the trajectory of the algorithm with a small η value, resulting in a slow convergence. On the other hand, if η is too big, there is a chance that the lack of precision prevents the algorithm from converging to the optimal value, provoking the method to *bounce over the walls of the well* (see figure 2.13b).

The behavior of the method of the steepest descent is illustrated in figure 2.13 for a one dimension problem, but there is no conceptual obstacle to apply it to multi-dimensional problems, since the gradient is a generalization of the notion of derivative, and a gradient can be computed for a function defined on a multi-dimensional domain.

Figure 2.13 — Three possible scenarios for the gradient descent algorithm looking for the minimum of a parabolic function. First (a) is optimal, second (b) does not converge, and third (c) converges slowly



a) Efficient exploration

b) Exploration lacking precision, failing to find the minimum

c) Exploration with excessive precision Very slow convergence towards the minimum

2.3.5.2 Trajectory-Based Optimization for Adaptive Gameplay

[Yannakakis and Hallam, 2009] have applied this optimization technique in the context of adaptive games. They organize existing works in two categories. The first category of approaches that seek to adapt games to users is the *implicit* category, where a system acts on several aspects of the games that are supposed to increase players' satisfaction, and then measures how well the game experience went. This type of work composes the majority of what has been described in this document so far. By contrast, these authors describe their own work as belonging to the second category, that is, *explicit* approaches. This means that the player's satisfaction is not measured, but instead is calculated by the adaptive system, which tries to optimize it. Therefore, the objective is to build a model of the player's satisfaction. The model considers opponents a player may face during a game session, and not other aspects of game design, such as game mechanics, game levels or narrative.

The context of this work is a classic game named *bug smasher* that consists of a 6×6 grid, large enough for a child to step on it, on which bugs appear at seemingly random times. The objective for the player is to smash bugs as they appear on the grid, that is, to step on the involved tile before the bug disappears.

The nature of a game session is characterized using Malone's factors [Malone, 1981], namely challenge and curiosity. The challenge can be either *low*, *average* or *high* depending on the speed at which the bugs appear and disappear, and the curiosity can also be either *low*, *average* or *high* depending on the *spatial diversity* of the game (i.e. the unpredictability of the localization of future bugs).

A set of 72 children played the game and reported their level of enjoyment. Depending on characteristics describing their playing style, such as response time, or movement speeds, they were sorted into 9 different classes. For each of these classes, a neural network is used to learn the model of the player. That is, this ANN is trained to predict the satisfaction level based on the nature of the game (described with Malone's factors).

Once the model is acquired, it is expressed in an analytical form in order to allow the calculation of partial derivative of the two criteria: spatial entropy and game speed. The gradient descent algorithm is applied to determine the appropriate modification of these criteria to enhance the computed enjoyment level.

The experiments conducted in this work showed no significant improvement of the enjoyment between a static game session and an adaptive game session, which is interesting considering that the algorithm in itself was not flawed. Reasons for that may involve the strict analytical description of the game dynamics, which may not suit the intrinsically chaotic human nature.

2.3.5.3 Analysis

Table 2.13 shows that algorithms like gradient descent can be seen as a useful alternative to more costly optimization techniques such as neural networks or evolutionary algorithms. Indeed, they are immensely faster than these, and they can be used to address problems that have highly evolving dynamics. When changes occur suddenly, and a system needs to

react quickly, gradient descent could be used to not waste time and quickly find an optimal solution again.

Moreover, these algorithms deal with the need of finding optimum to functions, which is abstract enough to express nearly any kind of concept. From objectives to constraints, from players resources to environment parameters, many design choices could benefit from such an optimization technique.

However to be applied on any given problem, there is a need for an analytical expression of the problem. As mentioned in the expressed requirements (see section 1.4), there may not be such an analytical description available. Besides, the fact that human individuals are involved in the system to be controlled – or optimized – makes it even harder to provide an analytical description of the system, since their dynamics need to be taken into account.

Table 2.13 — Gradient descent algorithms adequacy to the defined criteria for the game adaptation system

Characteristic	Adequacy	Comment
Genericity	++	Any aspect of gameplay could be dealt with
Ease of instantiation	−−	Needs an analytical expression of the whole game
Deals with great complexity	++	Needs an exhaustive analytical description
No strong assumptions on users	−−	Needs to model completely the player dynamics
Handles unforeseen situations	++	Reacts quickly to changes

2.3.6 Conclusion on Control and Optimization

Techniques described in this section offer interesting mechanisms to deal with the control of systems with a large variety of properties. There are a significant enhancements of classical control engineering when applied to target systems with a large search space, nonlinear dynamics, noise, randomness, or which change from one execution to the other.

It can be noted however, that there is always **a balance to be found between the amount of knowledge needed on a target system, and the ability of the controlling system to deal with evolution of the dynamics of the target system**. Biology-inspired techniques manage to control systems without needing *a priori* knowledge on its inner functioning. **They are however, highly sensitive to variation**, and they cannot adapt themselves if the very nature of the target systems changes. They would need additional time in a constant situation to find a new appropriate solution.

By contrast, some techniques can be resilient to change in the nature and properties of the system, but **they demand a significant amount of knowledge from the designer**, whether about the dynamics and behavior of the target system, or about the needed behavior of the control system.

Both of these advantages are needed to control the class of systems we are interested in, but **none of their drawbacks are acceptable**. For that reason, new paradigms have started to emerge to provide a more robust, more efficient, and more adaptive control of complex and dynamical systems.

2.4 New Paradigms: Distribution to Overcome Complexity

The problem of game adaptation to human players is vast, and can be addressed in many different ways. As we have seen all along previous sections, this problem has been addressed by different approaches, and even different fields. Whether a problem of machine learning, optimization, or pure design, the challenges are non trivial. Through all the proposed work we have listed in the previous sections, several aspects of the problem are being addressed, and many different concepts are considered.

But whatever the approach, there always seems to be a balance to be found between two aspects: genericity and complexity. It seems that the complexity can only be dealt with if a great amount of information about the system we are trying to learn, optimize or control is known. On the other hand, if only so few information is to be known about the system, then the complexity becomes an obstacle for the success of the adaptation system.

These reasons make us think that new paradigms are needed to efficiently and simply address the problems and challenges of such applications. It can be found in the literature that this challenge has been identified several times [Jennings, 2001]. An idea often mentioned to find a solution to such problems is to change the way we usually seek to solve them, and instead of trying to have a global view of the problem, we should try to use a bottom-up approach. Distinct parts of a system should be able to evolve autonomously and interact with each other. The result of their interaction should lead to a system of a higher order, that would be able to deal with the complexity of the task in a more efficient manner. In this section, we try to review several views of these ideas.

2.4.1 Agent and Multi-Agent Abstractions

In order to manipulate parallel dynamics, distributed sub-systems and interacting entities, a new abstraction has been advocated: the agent. Though there are many definitions of what agents are, as well as many use for this paradigm depending on the concerns of the people involved, a commonly accepted definition comes from [Weiss, 1999]:

"An agent is a computer system that is situated in some environment, and that is capable of autonomous actions in this environment in order to meet its design objectives."

Ferber [Ferber, 1999] takes it a step further, by adding a locality criterion: "An agent is an autonomous physical or virtual entity able to act (or communicate) in a given environment given local perceptions and partial knowledge. An agent acts in order to reach a local objective given its local competences."

These definitions set the emphasis on the fact that an agent should be acting in its environment, according to its own objective. The ability the agents have to modify their en-

vironment or to communicate with each other results in interesting phenomena at a higher level; when several agents are interacting with each other in a common environment, they are referred to as a *Multi-Agent System* (MAS).

A multi-agent system can be viewed as an abstract entity that exhibits its own behavior towards its respective environment. However this behavior is not explicitly designed but results from interactions of the agents composing it; therefore changing the organization of the agents changes the function of the MAS.

Agent and multi-agent abstractions have been extensively studied in the last decades, and many approaches in the field of control, among others, have designed agents along with means of interactions in order to obtain MAS with desired functions at the global level. When designing a MAS, no universal approach has been established yet, and when it comes to control, there are different possibilities, depending on what is distributed among the agents. In the following, three main approaches are presented.

2.4.2 Distribution of the Search Space Exploration: Swarm Algorithms

A first category of approaches that make use of the distribution to overcome problems inherent to the complexity of applications is inspired from organizational patterns found in certain animal species. These species exhibit groups behaviors that emerge from the interactions of individuals, without any central coordination mechanisms or strict hierarchy. The overall behavior of the group can therefore be considered as a bottom-up construction. Two of the main approaches using these principles are detailed and discussed thereafter.

2.4.2.1 Ant Colony Algorithms

A famous example of emerging group behavior is the ant colony. The cognitive abilities of ants are rather limited, and they do not posses sophisticated planning or reasoning skills. However, their cooperative behavior allows the colony to efficiently collect food in the environment. Initially, the ants wander randomly in their environment. When one of them encounters a source of food, it goes back to the nest, leaving behind a trail of pheromone. This is interpreted by other ants as an indication to where to find a food source. Consequently, other ants can find the source and get back to the nest. Since the pheromone trail evaporates over time, longer paths of pheromone levels tend to be less followed and are eventually forgotten. Similarly, when the food source is fully exploited, no pheromone is secreted by ants in the environment, and after a given period of time, the path is forgotten.

Ant Colony Optimization (ACO) algorithms make use of these principles to find solution of various problems [Colorni et al., 1991]. The key aspect of these approaches is to represent the problem as a graph, and solutions as paths within these graphs that reach a terminal node. Virtual ants are then deployed in the graphs and evolve together with behaviors inspired by those of natural ants to collectively discover the optimal path in the graph.

Each edge connecting two nodes i and j in the graph possesses two values: a pheromone value τ_{ij}, and a heuristic variable η_{ij} representing the *a priori* attractiveness of the edge. The pheromone level represents the accumulated knowledge of the algorithm, while the heuris-

tic value is obtained from domain knowledge as a way to improve the performances. An overall attractiveness of the edge is calculated based on these two values. The importance of each of them in the calculation are parameters of the algorithm.

At the beginning of the algorithm, ants are placed randomly on the graph. At each time step, each ant selects an edge to move within the graph. Each possible edge has a probability to be selected proportional to its attractiveness. This is repeated as many times as needed. When ants have all reached terminal nodes, that is, when they have all formed a complete solution, these solutions are evaluated with a global fitness function. The higher the fitness of the solution, the higher the pheromone level of the edges composing it is set. Similarly to its natural counterpart, the pheromone level in the algorithm steadily decreases over time to prevent the convergence toward suboptimal solutions. Ants are then replaced in the graph, and the process is repeated until a termination criterion is reached.

ACO algorithms have proven significant efficiency in various domains. From pure optimization, to scheduling, or resource allocation; but also in the field of control [van Ast et al., 2008] [López-Ibáñez et al., 2008]. Their nature is interesting for several reasons. They can deal with nonlinearities, and do not require analytical expressions of systems to be controlled. Besides, they exhibit better performances than other optimization techniques such as genetic algorithms [Hassan et al., 2005].

However, if one wants to use ACO for the control of a given system, there is still the need to express the control policies as graphs (see table 2.14). If the sequences of instructions of simple control policies can simply be expressed with the graph theory terminology, it may not be the case with modern video games, where the possibilities of control are virtually infinite. Moreover, for the algorithm to unfold correctly, solutions need to be evaluated a great number of times. In our case, evaluating a solution consists in letting the player play the game, and see if the experience was satisfying. We can easily understand that first, this can take a considerable amount of time, and additionally, that the evaluation function is not constant, that is, each evaluation of a solution influences the next evaluation. Thus, the same solution evaluated more that once would lead to different results.

Table 2.14 — Ant colony algorithms adequacy to the defined criteria for the game adaptation system

Characteristic	Adequacy	Comment
Genericity	++	Arbitrary control policies can be expressed through graphs
Ease of instantiation	+	Graph is a relatively simple way to represent domain concepts
Deals with great complexity	−−	Every single control choice needs to be expressed in the graph
No strong assumptions on users	++	No player modeling required
Handles unforeseen situations	−−	Needs time and iterations to find new solutions

2.4.2.2 Particle Swarm Optimization

Similarly to ants, the behavior of schools of fish and birds flocks is a good example of group behavior distributed in the individuals. In such groups of individuals, there is no central control or decision making. Each individual has its own perception of the environment, and makes decisions accordingly. While individuals are supposed to have only limited cognitive abilities, and behave in a mostly reactive way, at a higher level, it can be observed that the group behaves in a consistent way. The group has several interesting properties. No collisions between individuals are observed, and it can evolve to avoid predators, while preserving the cohesion between individuals.

Inspired by these group dynamics, Kennedy proposed an interesting method to explore a solution space in a distributed fashion, namely Particle Swarm Optimization (PSO) [Kennedy and Eberhart, 1995]. It falls into the category of swarm intelligence. Similarly to genetic algorithms (see section 2.3.3.1), Kennedy suggested that a population should be considered, in which each individual represents a solution of the problem called a *particle*. As opposed to GAs, in which new individuals are created from existing ones, in PSO, individuals evolve in the solution space, which is considered as their environment. Similarly to the behavior of previously described social animals, individuals – or agents – perceive their environment, and decide autonomously what their next move is going to be.

In PSO, the agents actions basically consist in moving within the solution space. At each point in time, each agent is able to evaluate the quality of the solution it represents. Therefore the objective of the agents is to move in the space in order to increase the quality of their solution, and thus, to find the optimal solution. The movement of each individual is however constrained by the state of the swarm. Each individual maintains not only its position, but also a velocity value, and the best position it has been on so far. At each time cycle, an agent exchanges information with its neighbors to determine the best known position of the neighborhood. Once this information is known, it updates its velocity, and finally, change its position before evaluating again the quality of the solution it represents.

Cooperation between solutions enables the algorithm to find better solutions faster than other optimization algorithms, such as Genetic Algorithms. Moreover, using such a population prevents the algorithm to fall in a local minimum, since a larger area of the space is covered by the particles.

The PSO algorithm can be applied in a large variety of control problems [Venayagamoorthy and Harley, 2007] [Boubaker and M'sahli, 2008]. No analytical description of the problem is necessary. One just needs to be able to describe the solution space to let particles evolve in it. Knowledge to apply this technique to a given domain should be easily obtainable, since the main information needed is an evaluation function of the solutions. As opposed to ACO, no particular formalism is needed, and the problem can be treated as is.

However, table 2.15 shows that these algorithms share the same weakness as genetic algorithms (see section 2.3.3.1): the evaluation of a solution still implies the actual testing of the solution with human beings. Given the number of evaluations needed by the algorithm, be it inferior to the one needed by GA, the presence of human beings in the evaluation makes it impossible to optimize the game content in real-time.

Table 2.15 — Particle swarm optimization adequacy

Characteristic	Adequacy	Comment
Genericity	++	No requirements on the expression of the solution
Ease of instantiation	++	Only needs a solution representation and a fitness function
Deals with great complexity	++	Each solution is evaluated on actual performances
No strong assumptions on users	++	No player modeling required
Handles unforeseen situations	−−	Needs time and iterations to find new solutions

2.4.3 Distribution of the System Model

The second category worth mentioning in our study that makes use of the agent and multi-agent concepts consists in modeling the system to be controlled or adapted. We already mentioned that an exhaustive and comprehensive modeling of systems such as modern video games is not always possible, due to the amount of co-existing entities along with the quantity of interactions that can be found among them. Therefore describing a model using separated autonomous entities, as well as defining their interactions is a possible way to overcome the complexity of the applications. Decentralized control allows the designer to focus of simpler problems.

Kubera et al. have chosen to build models based on interactions occurring in the domain [Kubera et al., 2011]. One strength of their approach is that agents are not only used to model only autonomous entities, but instead, every concept that can interact with its environment. Consequently, there is no struggle to determine what should be an agent, and what should be only a passive object.

These agents are described not by their behavior, but by the interactions they can have with each other. In order to ease the listing of these interactions, this work proposes software tools to create a matrix, in which one can state all these interactions. Each interaction can have preconditions, and also has a priority grade (the higher the grade, the higher the priority).

Modeling the domain like so, by declaring all the interactions in a static manner, makes the task reasonable for domain experts, as opposed to a complete algorithmic description of the entities.

Besides, the behaviors of the agents are not scripted, agents only execute their allowed interaction with the highest priority. If the environment of the agent changes, it does not prevent the agent from having a consistent behavior. The expert, or designer, does not need to enumerate all the possible situations in which agents may be, since the agents will adapt their behaviors to their environment.

This work is an example of distributed modeling of the domain that allows the complexity of the simulation to grow without affecting the difficulty of the description of the model.

This methodology has been successfully applied in various domains, such as serious games for training salesmen [Mathieu et al., 2013].

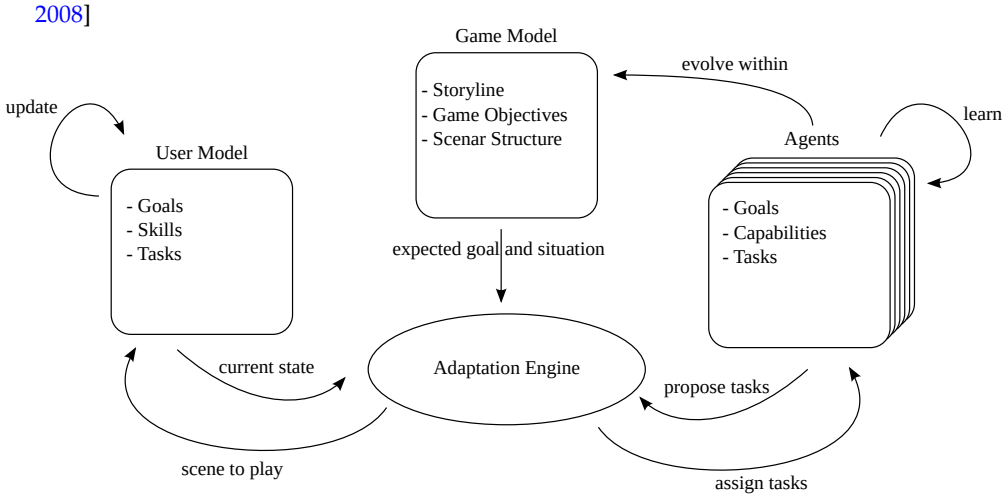
However, while the simulation itself may be adaptive, in the sense that agents' behaviors will properly deal with unexpected situations, the adaptation of the challenge level to the human is only done in an *ad hoc* way: a single performance value is measured, and in order to make it reach objectives, a single parameter, considered to be the only value that influences the difficulty is modified.

This mechanism may be inefficient in the case where multiple objectives are considered at once, and where more that one variable influence the difficulty of the challenges of the game.

To provide consistency along with adaptation mechanisms, Dignum et al. have proposed the use of an organization framework [Dignum et al., 2002]. An organization can be used to define objectives shared by individuals that take place in the organization. For that purpose, the concept of role is added. An individual can join an organization by occupying a specific role. The organization constrains the interactions that can occur between the various stakeholder. Various degrees of freedom can be given to individuals. From none, where everything is defined in the organization itself, to more, where only some landmarks are defined, and individuals determine their own actions. Such individuals – agents – can be implemented in any way, as long as they respect the contracts defined by their roles.

Westra et al. propose an organization model to be used in the context of games that require adaptation to users [Westra et al., 2009]. An overview of this organization can be seen in figure 2.14. The three main concepts used by the proposed adaptation engine are: (1) the storyline, (2) the agents, and (3) the user model.

Figure 2.14 — Agent Organization for adaptation in games, adapted from [Westra et al., 2008]



Agents are here used to represent entities that players have to interact with. Depending on the domain, they can represent players' colleagues, with which they have to cooperate to achieve specific tasks, they can also represent persons requiring the intervention of players, such as customers, or victims to be rescued, or they could represent enemies that the player has to defeat. An usual aspect of the proposed approach is that agents are also used to

represent any kind of dynamical aspect in a game, which cannot be qualified as *Non Playing Character*. This means that characteristics of the environment could be seen as agents: for instance the fire spreading rate in a firefighter simulation, the quality of weather in a flying simulation, etc. Each agent is capable of performing different actions, depending on its goal and the perception of its environment.

The storyline is defined as a sequence of scenes. Each scene represents a certain aspect of the domain. It can represent a specific sub-task needed to be learned by trainees, that can be characterized by the conditions of the environment, or the nature of the objective, or any other kind of constraints. During the game, one or several scenes are activated, and once they are done, their successors take their place. During the unfolding of scenes, agents are notified, and those who have plans that match the conditions of the scenes can act.

As the player interacts with the game within a scene, the measured performances are used to update the user model.

The role of the adaptation engine is to select the plans that should provide an optimal difficulty. The user model helps to determine which tasks would be more beneficial for the player. The interesting part is that each agent possesses its own preferences, along with knowledge about what it can or cannot do to remain believable. A combinatorial auction is set up to determine the plan to be chosen. If the difficulty needs to be adjusted, an agent is likely to modify its behavior, however, if it derives too much from what it believes to be consistent, then it no longer modifies its behavior.

The task of adapting the difficulty is distributed among the various elements that together form the content of the current sub-scene of the game. Similarly, each of these aspects guarantees that it is not modified in a way that ruins the believability of the game experience.

The complexity of the decision making process in a distributed fashion is then compared to a centralized equivalent. A centralized decision process would have to choose between the plans of all the agents for all the scenes. The conducted analysis found in the paper shows that the distribution diminishes the complexity by several orders of magnitude, and moreover, that a centralized process would be impractical with a total of four agents evolving during 30 scenes, which is a completely reasonable measure for a modern game.

Table 2.16 shows that this approach is quite efficient, and is able to solve problems that classical approaches cannot seem to handle properly. Complexity of real life games is a characteristic often left out by academic works, and distribution could be a solution to handle it.

However, the organizational framework described here implies that the design of the game itself is thought in terms of intelligent agents, scenes and sub-scenes, along with a proper user model. If there is clear advancement here, one may still wonder if it is possible to adapt a game that has already been designed, or that does not rely on plan-based implementations.

Table 2.16 — Agent organizations adequacy to the defined criteria for the game adaptation system

Characteristic	Adequacy	Comment
Genericity	−	Needs plan-based behaviors of all active entities
Ease of instantiation	++	Plans manipulate domain concepts
Deals with great complexity	++	Global solution is not designed, simpler sub-problems are considered
No strong assumptions on users	−	A user model is necessary
Handles unforeseen situations	++	Situations are adapted as they are encountered

2.4.4 Distribution of the Control Process

Another way of exploiting distributed computation consists in designing a solving process as a set of interacting subroutines. Though special care must be taken to lead the system to a satisfying state at the global level, it is possible to design adaptation and control mechanisms using a bottom-up approach.

Videau et al. have proposed a control system whose purpose is to control target systems on which only few information is available [Videau et al., 2011]. As opposed to the vast majority of the field of control engineering, the approach here does not make use of a model of the target system. Instead, the target system is considered as a black box, in which one or several outputs can be observed, and one or several inputs can be modified. The objective of the control process is to modify the inputs of the target system in order for the outputs to reach user-defined objectives. This approach has later been investigated by [Guivarch, 2014; Boes, 2014]

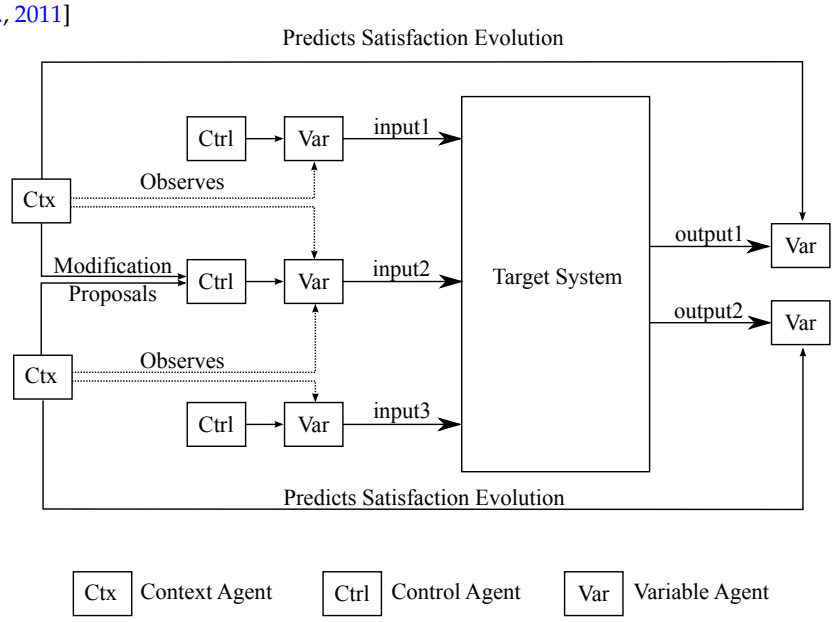
The bottom-up approach here consists in considering separately the task of controlling each input, as well as considering separately the task of observing each output. For that purpose, an autonomous agent named *variable agent* is created for each of these input and output values. Objectives expressed by users on specific outputs are given to their corresponding *variable agent*. Therefore such an agent is capable, in addition to observe the output value, to compute the satisfaction of the related objective.

The control policy is represented by a set of *context agents*. A context is defined as a set of input values, each one associated with a range of values. These ranges of values determine the required conditions for a context to be considered valid. When all the inputs of the target system are within the ranges of the context, then the context is considered valid. It is considered invalid otherwise. Each context possesses an action that can be applied on a given input of the target system. The action can be simple, like an atomic value modification, or it can be more complex, with for instance, a modification on an input value that lasts over time. Finally, each context is associated with a set of forecasts. A forecast is a prediction on the evolution of the satisfaction of a *variable agent*. In a nutshell, a context represents a particular state of the system, along with an action associated with the prediction of the consequence of this action.

Finally, *control agents* are responsible for actually modifying the values of the inputs of the target system. For each input of the target system, a *control agent* is created. A control agent responsible of a given variable receives the proposition of actions of valid contexts. If more than one proposition is submitted to a control agent, then it selects the best one by comparing the payoffs of the prediction of each proposition.

Figure 2.15 shows a static architecture of the described multi-agent system.

Figure 2.15 — A multi-agent system to control a target system as designed by [Videau et al., 2011]



At the creation of the system, contexts are created pseudo-randomly. Therefore, it is highly unlikely than the system is able to effectively control the target system. A learning process occurs at a local level. Each time an action is applied, and the consequences are not the ones that were expected, the forecast is adjusted. If at a given point, no interesting context is available, then one is created. Over time, it will be adjusted to be optimal. If several contexts are similar, they are merged together to prevent an explosion of the number of context agents.

The control process at the global level is never explicitly designed, nor is a representative model of the target system. By designing local agents with problem solving behavior, it is possible to focus on rather simple tasks, and to let emerge the solution at a global level. This multi-agent system is applied with success in the control of bio-processes, which are known to be complex systems, with nonlinearities, and that can only be modeled at high costs.

This work is interesting for several reasons (see table 2.17). First, it is quite general, as it does not make any assumptions on the nature of the system to be controlled, and can be applied to virtually any kind of system, as long as this system can be observed. Secondly, the computation time needed by the approach is low. The agents only exchange few information, and have a rather reactive behavior. The dynamics of the system to be controlled is indirectly learned through the context agents, but the inner functioning of the target system is not contained within them. They just represent the control policy to be applied. The

complexity of the target is handled thanks to the distributed aspect of the control system.

One could probably try to apply this kind of control system to a modern video game, however, there is still an aspect that remains unclear. Even if no particular knowledge is required to let the system find all the appropriate contexts along with their corresponding actions and forecasts functions, the amount of different situations in a game with several dozens of entities, each of them with their own parameters, is likely to generate a significantly large number of different situations. Moreover, in games like simulations, the variety among humans makes any adaptation system need the ability to quickly react to unforeseen situations. Only relying on situations that were previously encountered and properly dealt with may lead to a lot of inefficient game adaptation.

Table 2.17 — Distributed cooperative control adequacy

Characteristic	Adequacy	Comment
Genericity	+	Parameter-based policy is low-level enough
Ease of instantiation	++	Plans manipulate domain concepts
Deals with great complexity	++	Complexity of the target system is correctly handled
No strong assumptions on users	++	No user modeling required
Handles unforeseen situations	−	Needs time to handle never-seen-before situations

2.5 Conclusion

All along this section, we have studied artificial systems that have the ability to modify a game experience so it matches the needs of human players. Characteristics of the problem have led us to express additional requirements. The growing complexity of modern games implies that **a game adaptation system should be able to handle unforeseen changes in the dynamics without a precise model of the game system**. The diversity among human players **prevents us from making strong assumptions on their behaviors, desires, or dynamics**. The large variety of games that may benefit from such an adaptation system also implies that a candidate system **should not rely too heavily on specific game mechanisms**, in order to guarantee that the approach can be applied on various games. Table 2.18 is a a summary of all the studies we have conducted relatively to the criteria previously defined. From this analysis, we can observe that various fields have found interest in the problem of game experience adaptation.

Machine learning is often focused on the behaviors of artificial players or artificial entities evolving next to human players. The aim of most of these works is to adapt the behavior of artificial agents so the game experience is enhanced for the player. These approaches usually need to gather data from human beings playing the game in order to infer the best actions to take afterward. **Such information is not necessarily available, or too costly to gather**. Moreover, the adaptation through the changes in artificial entities behavior

is **focused on a specific subset of games, and is not applicable to a large set of games and simulations**.

Specific game design practices have tried to produce adaptive game experiences. However, **they heavily rely on specific game aspects**, limiting their interest in the general case. Moreover, they are usually **oversimplifying the problem, by reducing it to only a few dimensions**, where in reality there can be dozens of aspects that need to be taken into account. Finally, **they usually require classification of players in arbitrary categories that may not reflect the reality of actual people**.

Optimization and control theory also have been studied in the context of game adaptation. Work from classical control engineering, like PID controllers, **does not scale up to the complexity of games**, that may need the control of a large number of parameters. Techniques from artificial intelligence may be satisfactory in some way, but there is always a price to pay for them to work. **A balance needs to be found between the amount of knowledge needed by an approach, and the time it needs before it can handle a change in the dynamics**.

From these considerations, new paradigms have started to be used. Applications have started to be designed differently, and instead of consisting of a single block, they are designed as **distributed parts in constant interactions**. Optimization techniques have been using distribution with performance improvements. But it is multi-agent systems, with **the use of organization that have shown a real interest in the bottom-up approach**. The complexity is greatly reduced by designing autonomous agents interacting in the same environment. **Autonomous agents are also used to define a control policy independently of the domain**.

We observe that **no game adaptation system is fully satisfying at this time**. Machine learning or optimization always fall in the same pitfalls. Latest contributions seem to make use of the agent paradigm, whether by distributing the game elements, or by defining the control policy in autonomous parts. **This direction seems promising, as it leads to the greatest achievements**. We still however think that there is still work to be done to provide a better system or approach. In the next chapter, we present a theory along with certain tools, that will be used in the context of our contribution.

Table 2.18 — Analysis of the studied approaches according to our predefined criteria

	Genericity	Ease of instantiation	Deals with great complexity	No strong assumptions on users	Handles unforeseen situations
Q-Learning	−−	++	−−	++	−−
Dyna-Q	−	++	−−	++	−−
LCS	−	++	+	++	−−
LCS w/ ITS	++	−	++	−−	−−
Dynamic Scripting	−−	++	++	++	+
CBR	−−	++	−	++	=
User-Centered Design	++	++	−−	−−	−−
Scenario Rewriting	−−	++	++	+	−−
PID	++	−−	−−	+	−−
Experts Systems	++	++	−−	++	−−
GA	++	++	++	++	−−
ANN	++	−−	++	++	−−
Gradient	++	−−	++	−−	++
ACO	++	+	−−	++	−−
PSO	++	++	++	++	−−
Agents organizations	−	++	++	−	++
Cooperative control	+	++	++	++	−

3 Theory and Tools for the Study

In this chapter, we introduce a theoretical context, on which we are going to rely for the next part of our study. The exposed theory is associated with a methodology that exposes guidelines as well as best practices.

Motivations are first presented, then the theoretical principles are detailed before presenting the aforementioned methodology.

3.1 Limitations of our Design Abilities

The complexity and the dynamical aspects of the problem of adjusting a game experience to a human player have imposed severe limitations on the possible techniques and methods to be applied. This observation however, is not limited to the problem we are interested in.

A more general consideration is that artificial systems needed nowadays not only have more and more dynamical aspects, but they also tend to be open, with components that can enter the system or leave it without further notice. This is especially true with the increasing computing power and network abilities of devices, along with their always decreasing prices.

Modern artificial systems can consist of a large number of interacting entities, and tend to no longer be physically and logically centralized on single devices, or single pieces of software. The organization of all these interacting entities may no be known at the time of the design phase, and therefore no assumptions should be made on the topology of a system.

The functionality offered by a system is sometimes difficult to characterize, and the task of this system can be incompletely specified. It may be impossible to predict all the appropriate actions that the system should take. It can be considered to be non-linear, in the sense that a small variation on its input data may have, depending on the context, significant consequences on its outputs that were not previously observed for similar inputs.

For these reasons, the traditional approaches to solve problems or design artificial systems, that consist in successively dividing the problem into smaller sub-problems do not seem appropriate. Besides, the human abilities to consider and process all information and requirements are limited, and this tends to make these classical design approaches unpractical. Horn has introduced the concept of *autonomic computing* as a principle in which an

artificial system has the ability to autonomously modify its behavior, whether it is for reconfiguration or optimization purposes [Horn, 2001]. These abilities allow an artificial system to adapt to unforeseen situations and to handle large quantities of information.

A proposed way to achieve effective autonomic computing is to not focus on the global task to achieve, but instead on smaller, simpler entities which interactions will lead to an emerging global behavior.

3.2 Emerging Phenomena

Emergence is not a trivial notion, and it has been discussed since the ancient history. Differentiating what is emergent from what is not is by no means simple, but it is accepted that a phenomenon can be considered emergent when it cannot be simply decomposed. When observing its parts separately, one cannot generally deduce what happens at the global level [Golstein, 1999].

Typical examples can be found in social animals. When considering single ants, we observe a rather simple and reactive behavior, that consists in moving randomly until food or pheromone is found. Ants possess no strong cognitive abilities nor planning skills. However, when many individuals are regrouped to form a colony, the colony as a whole exhibits complex strategies. A colony is able to find the shortest path to food, and can resist to sudden changes in the environment [Goss et al., 1989].

Similarly the bird flocks are a notable example of emergence found in the nature. Birds are simple and do not have a sophisticated behavior. When evolving in large groups distinctive patterns can be observed, which cannot be deduced by the study of individuals[1].

An interesting point is that if these emerging phenomena cannot be designed in a centralized manner, by traditional problem decomposition, since they feature a virtually infinite number of sub-problems to solve. In the case of emergence, the phenomenon is the occurrence of patterns that arise from self-organized of simpler entities.

Consequently, there has been a growing interest about emergence in artificial systems [Stephan, 1998; Ronald et al., 1999]; the idea is that a system can be simple to design, and still offer a complex functionality, provided that it emerges from the designed entities. The difficulty then lies in finding the right approach to design simple, locally interacting entities that let emerge the desired phenomenon. For that purpose, the multi-agent system paradigm seems adequate, since it allows a separation of concerns, and puts the focus on simpler entities in interaction with their environment.

3.3 Multi-Agent Systems

From the desire to design locally interacting entities in a common environment, multi-agent systems (MAS) appeared to be offering adequate concepts for both modeling and implementation concerns. As mentioned in section 2.4, an agent is an autonomous entity that

[1] http://epod.usra.edu/blog/2006/06/black-sun-in-denmark.html

has a set of abilities and/or skills and that can locally perceive its environment. Given its perceptions, it adopts a behavior that leads it to take actions in its environment.

In multi-agent systems, several agents are put in a common environment. Therefore the environment perceived by a single agent can include parts of the environment of the MAS as a whole, but also other agents. These other agents perceived by a given agent are called its *neighbors*.

Multi-agent systems are used in many different ways, and some approaches do not seek to produce emergent phenomena. The MAS paradigm is often used to model complex interactions between few individuals with strong cognitive abilities. It can also be used to effectively implement physically distributed programs to encapsulate the logic of the interactions over the network [Russell et al., 1996].

In other cases, the designed agents are more reactive, and the interesting property is what emerges from their interactions. *"One of the main motivations to employ autonomous agents is the idea of emergence. Autonomous agents, by definition, behave in the real world without human intervention. One of the fascinating features of autonomous agents is that they exhibit so-called emergent behaviors, that is, behaviors not programmed in the agents by the designer."* [Pfeifer and Scheier, 2001]. Giving agents an autonomous behavior can lead to emerging behavior of the system. The next challenge when trying to design an artificial system with an emergent functionality, is to design agents in order for the functionality to be adequate, that is, that behaves the way designers intended it to. Indeed the sole fact that the global functionality emerges from the local interactions of the agent is not a guarantee that it is satisfying. The Adaptive Multi-Agent Systems theory is a solution to answer this challenge.

3.4 Adaptive Multi-Agent Systems

The goal of the Adaptive Multi-Agent Systems (AMAS) theory is to design and implement artificial systems where the global functionality emerges from the interactions of its constitutive parts: the agents. It seeks to build systems that are functionally adequate with regards to the designers' requirements, when no solution is known from designers, and when the task that needs to be done by the system is incompletely specified. Adaptive Multi-Agent Systems are resilient to changes in both their environment and their inner organization thanks to the self-organization of their constituents which does not rely on the effectiveness of the global function [Capera et al., 2003; Georgé et al., 2004].

The theory defines the *cooperation* as the engine of the self-organization of the agents composing the system [Gleizes et al., 1999].

3.4.1 Functional Adequacy

The theory states that interactions between a system and the environment in which it is situated can be of three different types:

- *cooperative* when the system and its environment benefit from the activity of each other, and maintain their activity;

- *indifferent* when the actions have no influence whatsoever on each other activities;
- *antinomic* when the interactions prevent the system from maintaining its activity.

More intuitively, this means that the activity found in a system environment can either be beneficial for the agent, it can have no consequences, or it can be a negative disturbance. Any interaction falls in one of these three categories [Camps, 1998].

The notion of functional adequacy is introduced by the theory. A system is said to be functionally adequate when it interacts with its environment in the way it was intended to.

Given these definitions, [Glize, 2001] expresses the following theorem : *"Given a functionally adequate system, there exists at least one cooperative internal medium system that fulfills an equivalent function in the same environment."*

A *cooperative internal medium system* is a system in which all the interactions between its constituting parts are *cooperative*

Therefore, considering this theorem, to design functionally adequate systems, it is needed to focus only on the local interactions of their constituting parts, and to make sure that they only have cooperative interactions. This confers the advantage that it is not needed to verify the global function of the system, but only to focus on local interactions. If the local interactions are actually cooperative, then the global system is functionally adequate.

3.4.2 Adapt the System through its Parts

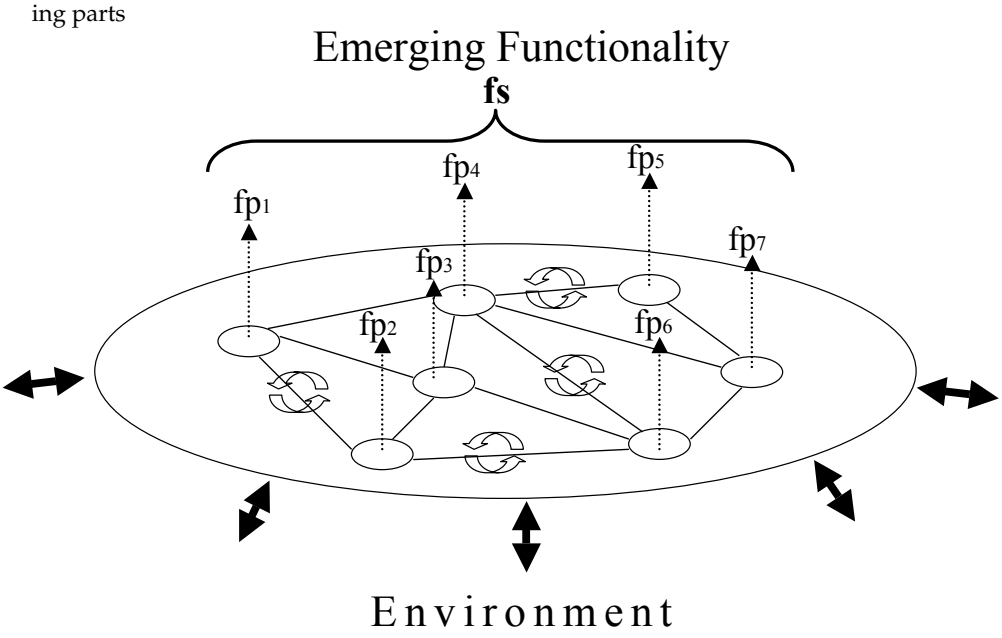
As soon as a change occurs in the environment, then the functionality of the system may not be adequate anymore. If the function is no longer adequate, it means that relations that are non-cooperative exist between the constituting parts of the system. In order for the system to be able to adapt to changes in its environment, agents composing the system need to be able to locally detect that they are facing non-cooperative interactions. When they figure out these non-cooperative interactions, they should be given means of restoring the cooperative nature of their interactions.

If agents can locally restore the cooperative nature of their interactions with their own environment, then the global system is again in a state in which it only has cooperative interactions with its environment, and therefore, it is again functionally adequate.

In that sense, an Adaptive Multi-Agent System functionality is emergent, since it is not explicitly designed, and the way it can face disturbances in its environment is not predicted in advance, but instead, the local behaviors of its constituting parts let emerge an adequate and adaptive functionality.

Each constituing part of the system has partial functionality fp_i, and is in interaction with a limited part of the global system environment. The interactions between these partial functionalities constitute the global function fs. By changing the interactions of the constituting parts of the system, the local functions fp_i are changed, and as a consequence, the global function f_p is modified. In other words, changing the global function consists in locally changing the interactions that each partial function fp_i has with its own environment (see figure 3.1).

Figure 3.1 — Emergence of the system functionality thanks to the activity of its constituting parts



In the case of our problem, using this paradigm could be of a great help (see section 4.3). In order to design such adaptive MAS, a methodology has been proposed to guide the unfamiliar system designer. The next section describes this methodology.

3.5 ADELFE: A Methodology to Design AMAS

In order to use an Adaptive Multi-Agent System, one must specify the agents composing the system, along with their behaviors, the characterization of what makes their interactions cooperative or not, and how to restore a cooperative state if needed. This is far from trivial, and a methodology has been created specifically for assisting designers.

ADELFE stands for *Atelier de Développement de Logiciels à Fonctionnalité Émergente*, which could be translated to *Development Toolkit for Software with Emergent Functionality*. It is based on the well-known software development methodology *Rational Unified Process* (RUP).[Picard, 2004; Bernon et al., 2005; Rougemaille, 2008]

Since developing an AMAS still consists in developing software, ADELFE has kept most of the RUP content; for instance, the phases where requirements must be elicited, or where analysis are conducted, before finally going through the implementation phase. But since the design and deployment of an AMAS differ from those of traditional systems, several steps were added in the process in order to cover these specific needs, such as determining if the problem needs the AMAS approach, identifying the agents, and finally determining how the notion of cooperation is integrated into the various parts of the system.

An overview of the methodology is shown in figure 3.2, from the early analysis, to the final implementation. It consists mainly of four different phases. The first phase is very similar to traditional software engineering methodologies. It consists in rigorously defining

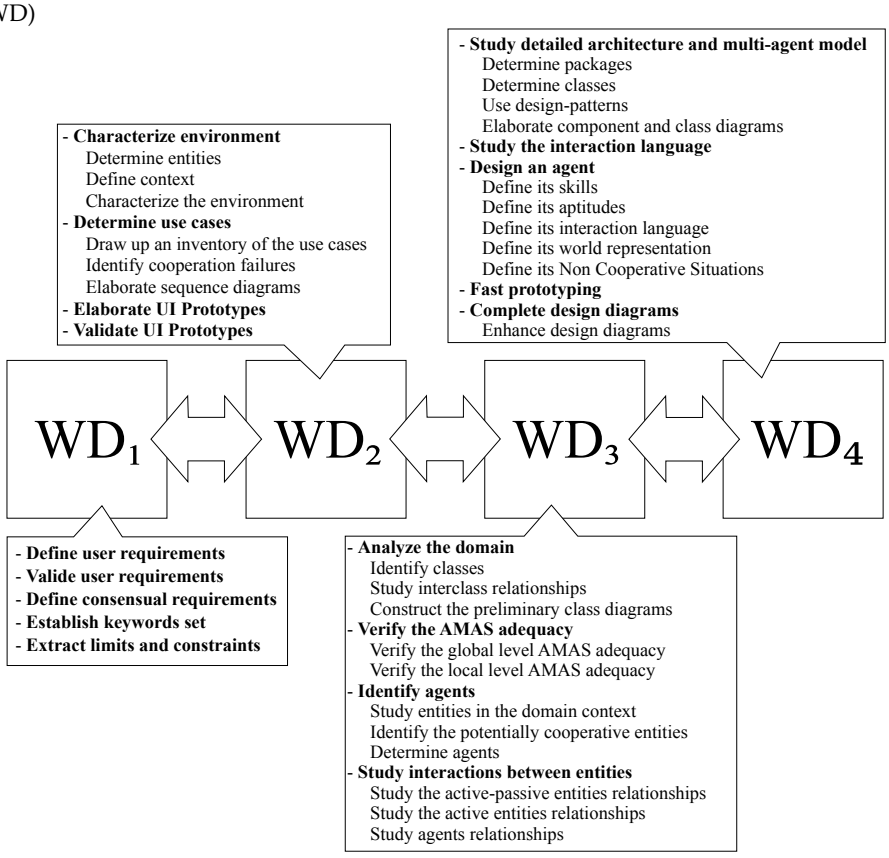
the requirements we have on the system, as well as clearly defining the scope of the system.

The second phase analyzes the environment in which the system is located. The various entities identified in this environment play a key role in the design of the system itself.

The third phase contains many additions relative to the AMAS theory. It is in that phase that, for instance, we must determine if the approach is appropriate for the problem as it has been analyzed so far. Agents will also be identified in that phase, and their interactions with their environment will be described.

The last phase involves the design of the agents in terms of implementation. Behaviors are detailed, and finally the system can be implemented. The next chapter go through these phases in order to design an adaptive multi-agent system able to dynamically adapt a game experience to players.

Figure 3.2 — Overview of the ADELFE methodology, broken into four Work Definition (WD)



Part II

Contribution Design and Realization

4 Requirements and Analysis

In order to build an artificial system whose functionality is to adapt games experiences to users, we need to provide precise requirements, in addition to those described in chapter 1. The ADELFE methodology (see section 3.5) guides us through the different steps of requirements elicitation. In a second time, further analysis is needed to verify the adequacy of the AMAS approach on the one hand, and to guide us in the first steps of the design phase on the other hand. These steps enable us to provide an adequate design for the game adaptation system.

4.1 Final Requirements

We start by describing the environment in which the system evolves at runtime, mainly by describing the entities that can be found in it as well as their interactions, and finally, we characterize this environment using a set of criteria defined by the methodology.

4.1.1 System Environment

4.1.1.1 Entities

The environment of the system is defined as a set of entities that interact with the system we design. We distinguish here two types of entities.

Active Entities. They have their own dynamics. They can initiate activity even in the absence of external stimuli. Two types of active entities are identified in the environment of the system:

- A *human expert*. Depending on the context, this expert can be different persons. In the case of a serious game, that can be a trainer or a pedagogical expert, who initiates the game experience and is able to judge what is needed for trainees. In the case of a pure entertaining game, that person could be a game designer that expresses requirements about what a game experience should be like. In both cases, the expert designs game scenarios to be played by players.
- The *game platform* on which the game is implemented. Scenarios unfold on it and players need to manipulate it to experience the game. Therefore players are not directly

perceived by the system we want to design, since they are only perceived through the game platform. The game platform in itself can trigger activity, depending on how it has been designed, with, for instance, randomly generated events.

Passive Entities. They are present in the environment, but they have no dynamics of their own. They can only be perceived by active entities and the system itself, and be potentially altered by them. In our case, we identify a single passive entity in the environment of our system:

- The *game scenario* which contains information on how a certain game session should unfold. It consists of a parametrization of the game engine, that defines the challenges presented to the player, along with the conditions in which they have to be realized. The game scenario is created by game designers and/or pedagogical experts, it is consumed by the game engine, and through it, experienced by the players.

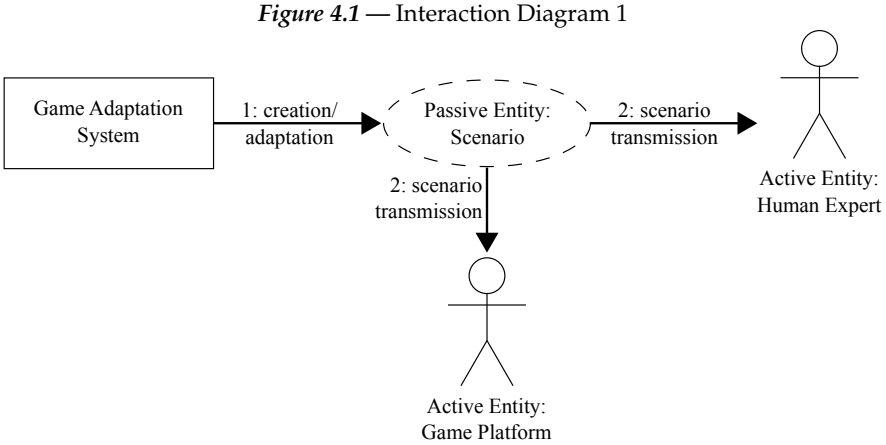
4.1.1.2 Interactions

Several interactions exist between entities in the environment and the game adaptation system. They can be expressed through interactions diagrams, in order to minimize any ambiguity resulting from a simple textual description.

We first identify two different flows that are exchanged through the only passive entity found in the environment.

- The first interaction, depicted in figure 4.1, shows that the game adaptation system creates, or adapts, a scenario to make it suitable to the player, and then sends it to the game platform. That scenario may also be sent to experts, since they may want to modify it.

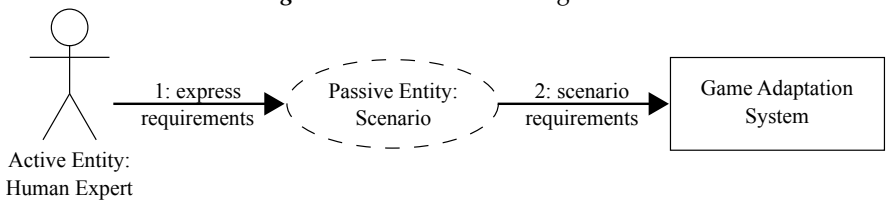
Figure 4.1 — Interaction Diagram 1



- The next interaction we describe, shown in figure 4.2, allows the human expert to transmit the characterization and the requirements of the scenario to the game adaptation system. Indeed, to properly define an adequate scenario, the game adaptation system

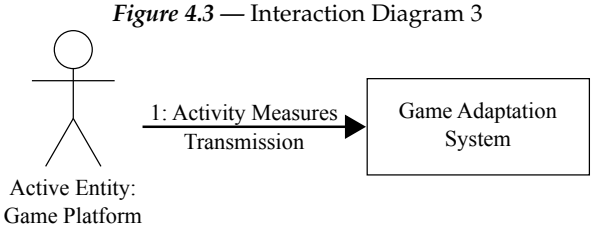
needs to be aware of what is expected by the experts or designers regarding the content of the scenario.

Figure 4.2 — Interaction Diagram 2



The last interaction diagram describes the information flow that results from the interaction of the player and the game platform. After the game platform is parametrized with a supposedly adapted scenario, the player interacts with the various artifacts in the game, and seeks to reach the predefined objectives. The player activity is monitored, and then sent to the game adaptation system in order to determine if the game experience is adequate or not (see figure 4.3).

Figure 4.3 — Interaction Diagram 3



4.1.1.3 Characterization

ADELFE then proposes to analyze the environment of the game adaptation system using the characteristics defined by Russel and Norvig, as they are appropriate dimensions to define difficulties encountered by artificial systems [Russell et al., 1996]:

- The environment is *dynamic*: in our case, the entities present in the system environment are modified by the system itself, but they are also modified by other means. The two active entities can modify their own state by themselves, but as described in the previous sections, the passive entity "scenario" can also be modified by the two active entities. Therefore modifications in the environment can occur even when the game adaptation system does nothing significant.
- The environment is *continuous*: the artificial system is likely to manipulate continuous values in the game scenario. Additionally, provided that such mechanisms are available, nothing prevents the game adaptation system from observing what happens in the game engine in real time, dealing once again with continuous values.
- The environment is *non-deterministic*: stochastic aspects could potentially be found in the game engine, depending on how it has been implemented. But even if the game is purely deterministic, there are still interactions coming from the players which we are

going to consider stochastic here. Therefore, whatever the game we are interested in, the environment of the game adaptation system is non-deterministic.

- The environment is *non-accessible*: not all information that could be used is available to the system. The most clear example is the inner state of the player. It could be used to determine the best game experience to provide. It is not possible for the game adaptation engine to simply predict the results that will be obtained for a given challenge.

This analysis reinforces the idea that the problem is difficult and that there is an actual need for an intelligent system to provide mechanisms to adapt the game experience to players.

4.1.2 Elicitation of Use Cases

In order to have a more precise idea on how the system will behave towards its environment, we then need to elicit the use cases, and then determine what a failure could consist in.

4.1.2.1 Use Cases List

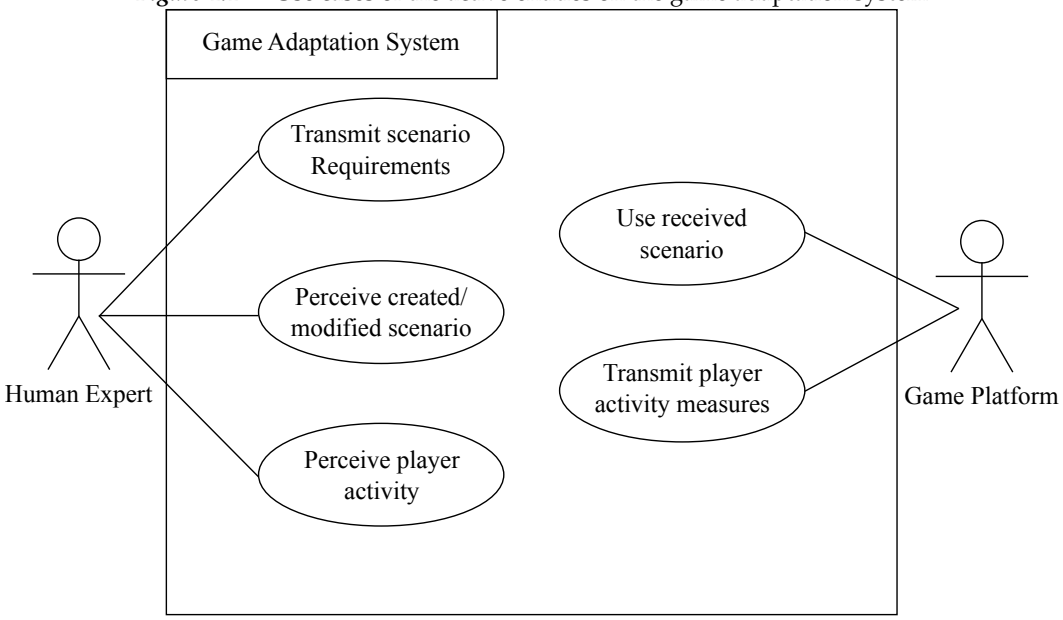
For all active entities in the environment, we list the use cases every active entity in the environment may be in when interacting with the game adaptation system. We distinguish the use cases of the human expert, that involve the manipulation of the scenario, and the use cases of the game platform (they all can be seen in figure 4.4):

- The human expert should be able to:
  - express requirements on the game scenario, that define the scope and the objectives of this scenario,
  - perceive the scenario tailored by the game adaptation system,
  - perceive the activity of the player on the game platform.
- The game platform (with which the player interacts) should be able to:
  - receive a scenario as it is created and/or modified by the game adaptation system,
  - transmit information about what the player is doing in the game to the game adaptation system, in order to determine the proper actions to be taken on the scenario for offering a better game experience.

4.1.2.2 Cooperation Failures

From the listed use cases, we identify those presenting a risk of cooperation failure. A cooperation failure occurs when the system no longer has cooperative interactions with its environment, and therefore, is not functionally adequate. Some situations cannot be corrected by the designer in a straightforward manner, since they are not just software errors,

Figure 4.4 — Use cases of the active entities on the game adaptation system



but instead, they denote cases in which the high-level behavior of the system needs to be modified.

There are two use-cases in which cooperative failures may occur. When a human expert visualizes the scenario that is being used by the game platform, this scenario may not be consistent with the previously expressed constraints and boundaries. This may be the case if the game adaptation system has modified the scenario in a way that goes against the activity of the human expert. In that case a cooperation failure occurs between the system and its environment, in that case, the human expert.

Another possibility of cooperation failure may occur when the game platform receives a new scenario. It is possible that the content of the scenario prevents the player, through the game platform, from reaching the objectives defined by designers and/or experts. In this case, the activity of the system that has modified the content of the scenario and the activity of the game platform are antinomic.

These two failures need to be remembered, as they will later be used to determine proper actions to be taken by smaller parts of the system to be designed.

The final step to be taken, before starting to think in terms of agents, is to analyze the entities relatively to the domain, that is needed first to validate the adequacy of the approach, and then to identify the best agent modeling.

4.2 Domain Analysis

Given the various interactions between the environment entities and the game adaptation system, we need to provide a more accurate analysis of the domain, that will be used in the design of our solution. Each entity must be described along with its means of interactions. If an entity is too abstract, it needs to be decomposed into simpler sub-concepts, easier to

analyze.

The first entity we are interested in is the game adaptation system itself. We saw in the interactions described in figure 4.1 that the game platform needs to receive information about the newly adapted game scenario. In the interactions we have described, the game adaptation system is an entity of high granularity. Consequently, we need to decompose it. We consider the game adaptation system interactions separately for each aspect of the game that can be modified. Since we want to be as generic as possible, we should not make any assumptions on how a specific game works and what specific game artifacts it has introduced.

4.2.1 Game Adaptation Information Transmission: Parameters

As shown in the first use case in figure 4.4, the game adaptation system transmits to the game platform a new game experience that will supposedly be better for the current player. Depending on the game, the "new experience" can be expressed in various forms, whether it is plans, stories, characters and so on. The concern of being as generic as possible leads us to think with concepts of the lowest possible level of abstraction.

For that reason, we propose to think in terms of numerical values, that can represent any aspect of a higher level game experience: a *parameter*. A *parameter* can represent any characteristic of a game artifact, for instance the speed of a moving entity, the effectiveness of a certain action, the quantity of an available resource, the consumption rate of another one, etc.

A parameter is characterized by a name, used to identify it in a unique way. Each parameter has a definition domain. That definition domain is arbitrary, and is defined by game designers: for example the set of real numbers, or positive numbers, or any other subset. Modifying the value of one parameter changes the game experience by modifying the way the game behaves.

Since each parameter represents a certain aspect in the game experience, for each game session experienced by a player, there is a corresponding scenario that contains all the relevant parameters. The set of parameters can be considered as the *input* of the game. More formally, for a sequence of the game S, the scenario has an input $IN(S)$ defined as:

$$IN(S) = \{p_1, ..., p_m\}; p_i \in [p_{i_{min}}, p_{i_{max}}]$$

Designing a game consists in creating the game artifacts, and defining the corresponding parameters and domain definitions. However creating a scenario for a particular game consists in setting values for these parameters, and by doing so, creating a specific game experience. Therefore in the second use case defined in 4.1.2, the modification of a game scenario consists in specifying the values of all parameters defined for a given game experience.

In the next section, we analyze the interaction described in figure 4.3, that is, when the game platform must send information about the player activity. The game platform needs to be observed by the game adaptation system. Since the game platform itself is also a complex entity, we need to decompose it, and consider its constituting parts separately.

4.2.2 Game Platform Representation: Measures

In order for the game adaptation system to have a view on the game platform that is as generic as possible, we must avoid the use of concepts relative to a particular game genre. Similarly to the scenario description presented in the previous section, we consider simple numerical values to be of the lowest level of abstraction possible. Virtually any kind of game could translate its own concepts into numerical values.

To reduce the complexity of the video game platform, we decompose it into simpler concepts. We define the representation of the player in interaction with the game platform as a set of measures. When playing, there are interactions between the various artifacts that can be found in the game and the player, though the input of the platform. While challenges are presented, and the player tries to complete them, the activity on the game platform can be observed, or measured by the game engine. The nature of each measure depends on the game domain, but whatever the game genre, virtually all games feature such measures. It is up to the game designer to determine the relevant measures. Examples are a success rate for a given activity, the time a player has taken to reach a certain achievement, or even the resources a player has consumed over time. Depending on the imagination of game designers, the list of possible measures on a game is infinite.

Similarly to parameters, all measures are identified by a name. We associate each measure with a range in order to contextualize its value. All measures are not necessarily relevant at the same time. Therefore for a given game sequence S, the perceived activity of the player is given by the game platform as a set of values $OUT(S)$ defined as:

$$OUT(S) = \{o_1, ..., o_n\}; o_i \in [o_{i_{min}}, o_{i_{max}}]$$

In the next section, we analyze the last active entity, that is, the human expert, who, as described in figure 4.2, has interactions with the game adaptation system, in order to define the scope of the game experience, as well as what is expected from the player.

Similarly to the two previous descriptions, we need to decompose the entity in order to deal with simpler and generic concepts.

4.2.3 Human Experts Requirements: Objectives and Constraints

Before any adaptation can occur, the human expert needs to express the way the game should unfold, and the goal the player should reach. The expert knows what is relevant for the underlying entertainment or pedagogical objectives, and the adaptation to the player should be done considering this knowledge. The expert is present in the game adaptation system through this information. To deal with it in a simpler way, we choose to decompose it into simpler concepts.

The first type of requirement from the expert is that players are facing at all time challenges that match their abilities. Therefore we need concepts to express the nature of the challenges presented to the players.

We consider challenges as objectives to be attained. Objectives can be defined in an entertainment context, or they can be pedagogical objectives. By referring to the Bloom's

taxonomy [Bloom et al., 1956], an objective is defined as a task to conduct, with a measurable performance in specific conditions.

Tools are needed to define these two aspects. They are conditions and performances. The conditions in which the player is are defined by the values of the parameters of the scenario, as they can potentially define every aspect of the game environment. Performances of players can be observed through *measures* of their activity, as defined in the previous section.

Therefore, in order to specify an objective, a game designer defines:

- constraints on a set of parameters of the scenario: they are authorized values for parameters that define the conditions in which the performance needs to be done;
- objectives on measures of the game: they are target values that define the actual performance required from the player.

This information is required from human experts (game designers or pedagogical experts), and can be difficult to express. Most of the time, experts have an intuitive knowledge or approximate requirements which are not easily expressed in precise numerical values. For that purpose, we introduce satisfaction functions.

Satisfaction Functions – Criticality Functions

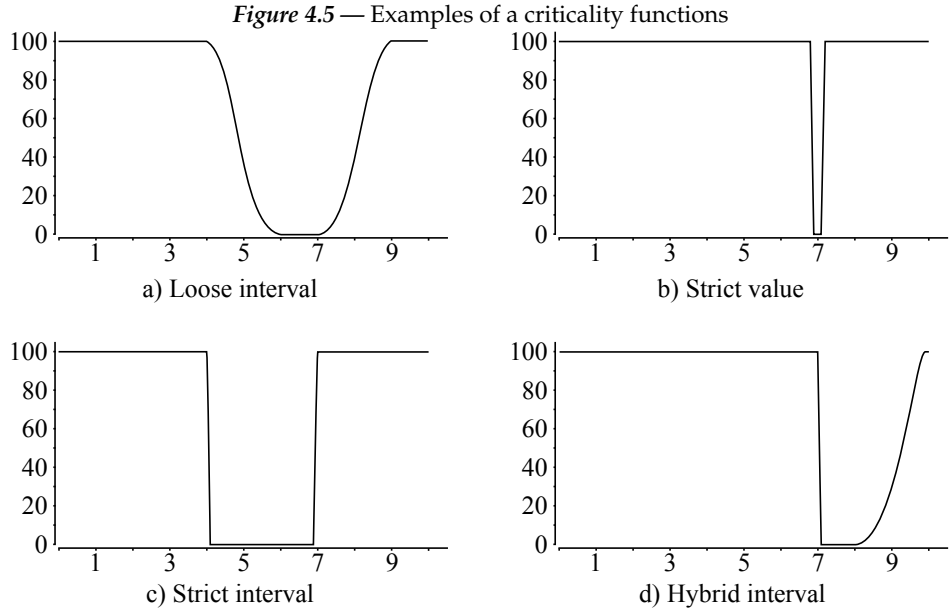
Satisfaction functions are used to express the degree of satisfaction of a constraint or an objective. These functions map the value of a variable to a satisfaction value. The satisfaction value is arbitrarily normalized to always be comprised between 0 and 100. We propose to use them to express both constraints on parameters, and objectives on measures. Similarly to fuzzy predicates, we consider, with satisfaction functions, that a requirement can take any value ranging from completely satisfied to completely unsatisfied.

In order to work with the AMAS theory, we need to apply a transformation on the concept of satisfaction function. The theory introduces the notion of criticality. The criticality is a value that expresses to what extent an agent is in an undesirable situation. In other words, it expresses how much the agent is *unsatisfied*. Consequently, the transformation from *satisfaction* to *criticality* is quite simple. The *criticality* is defined as the complementary value of the satisfaction: a minimal (0) criticality implies a completely satisfied objective or constraint, and a maximal criticality (100) characterizes a completely unsatisfied objective or constraint. Therefore from now on, we are going to refer to *criticality functions* instead of *satisfaction functions*.

From a designer perspective, we propose to consider criticality functions in an intuitive way, with graphic representations. See the four examples in figure 4.5. The functions can represent constraints on parameters: the null criticality values represent the value range where the parameter should be (for instance on figure 4.5*a*, between 6 and 7), and the high criticality values represent value ranges that are not acceptable for the parameter (for instance on figure 4.5*a*), lower than 5, or greater than 9). Similarly, for objectives on performance measures, null criticality values indicate that the objective is reached, whereas a high criticality level indicates a complete failure.

When considering a graphical representation of the criticality function, the knowledge and intuition of human experts can be expressed through the *shapes* of the curves. A wide curve, as shown on figure 4.5*a* can express a relatively loose constraint, where an interval is defined ([5,7]), but where the criticality is not maximal as soon as you get outside this interval, but instead gradually goes up as the value changes. A narrow curve specifies a precise value, for instance 7 on figure 4.5*b*. Additionally, very specific objectives ranges can be expressed (figure 4.5*c*), or by contrast, others can have custom borders (figure 4.5*d*).

Figure 4.5 — Examples of a criticality functions



To actually express and manipulate these criticality functions, we propose to use an abstract function that can be parametrized. The parameters represent the points where the function starts to decrease, where it is null, where it starts to increase, and when it reaches its maximal value. No particular knowledge is required to define such a function, as only its aspect is used to express a requirements, as shown in the previous example. For more details about how these functions are actually defined, see the annex A.

4.3 AMAS Adequacy

With the ADELFE methodology comes a tool which purpose is to guide the designer during the analysis of the environment, in order to determine if the AMAS approach is adapted to the problem or not. It consists of a set of questions that characterize the problem, and according to the answers provided by the designers, a recommendation is provided by the tool.

We expose here the questions, and detail the answer for each of them before presenting the results yielded by the tool.

Is the global task incompletely specified? Is an algorithm a priori unknown? There is no known a priori knowledge to determine what a proper game adaptation is going to be,

since all players may be different. Additionally, the genericity requirement prevents us from making any assumption on the nature of the task players will have to go through, making difficult any specification on the solving process.

If several entities are required to solve the global task, do they need to act in a certain order? The solving process is likely to be the result of the coordination of the measures of players activities and the subsequent modifications of the game scenario. Since both activities and scenario modifications are decomposed into several entities, orders in their behaviors are likely to have an influence on the quality of the global solution.

Is the solution generally obtained by repetitive tests, are different attempts required before finding a solution? Since we have stated in section 1.4 that no assumptions should be made on players, it is really unlikely that a single iteration of the adaptation process is going to guarantee the satisfaction of the human expert requirements. For that reason, several iterations are probably going to be necessary before finding a satisfying solution.

Can the system environment evolve? Is it dynamic? The environment of the system is, among other things, composed by a game platform with which interacts a human player. We already mentioned in 4.1.1.3 that the system environment is non-deterministic, but the indirect presence of the human makes it highly dynamic. Humans evolve over time, and are subject to frequent changes, and besides, these changes may not be correlated to what happens in the game. Therefore, even if the system is in a perfectly stable state, its environment is subject to spontaneous changes.

Is the system process functionally or physically distributed? Are several physically distributed entities needed to solve the global task? Or is a conceptual distribution needed? A priori, the system does not need to be physically or logically distributed.

Is a great number of entities needed? The decomposition proposed in section 4.2 implies that, depending on the game, many different entities are involved. Modern video games model an increasing number of aspects to build their gameplay, and all these aspects can be subject to modifications to propose an enhanced game experience. Moreover, challenges proposed to players can require various skills, and consequently, increase the number of significant measures on the player activity.

Is the studied system nonlinear? The system does not satisfy the principle of superposition. In a given state, an action A may have a consequence α on the system, and an action B may have a consequence β, it does not imply that the conjunction of actions A and B has, as a consequence, the conjunction of results α and β. The system is nonlinear, since, depending on the context, similar actions may have largely different consequences. Similarly, in a given state, a slight modification on the input of the system, may have significant consequences, which were not previously observed.

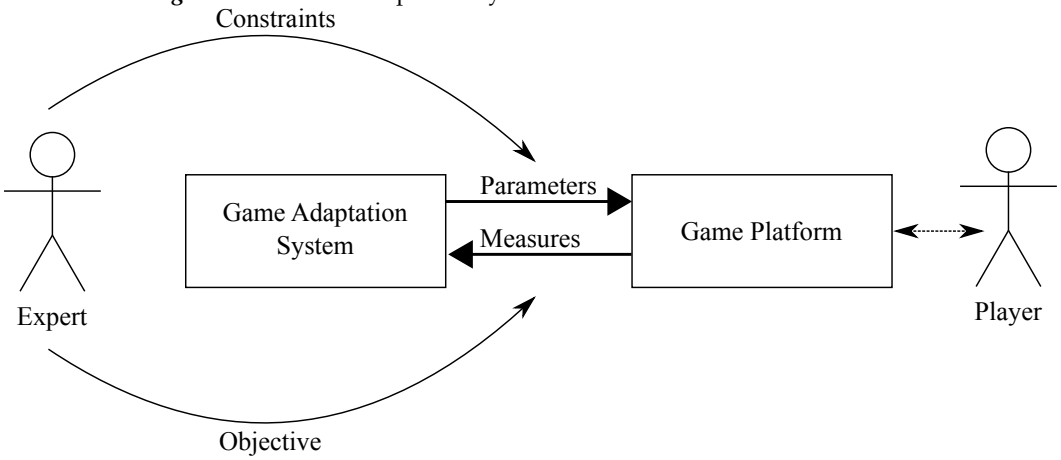
Finally, is the system evolutionary or open? Can new entities appear or disappear dynamically? It is possible that human experts express new requirements in a dynamical fashion, however, this is not our main concern, and for the major part, the population of entities present in the system is going to remain constant.

The majority of answers to the questions makes us think, with the help of the methodology tool, that the AMAS approach is an adequate approach for the problem as we have described it.

The goal of the game adaptation system can now be expressed in a more formal way. As depicted in figure 4.6, in order to provide an optimal game experience, through the proposition of adapted objectives to any player, the system should be able to determine a set of values for all parameters that

- maximize the satisfaction of all the constraints expressed by game designers or pedagogical experts, and
- make the player maximize the satisfaction of all the objectives that are relevant during the current game session.

Figure 4.6 — Game adaptation system interactions with its environment



This section concludes the environment description and the requirements analysis phases of the ADELFE methodology. Now that we have a better view on the problem, and that we have verified the adequacy of the approach, we can proceed to the actual design of the system. This design consists of a bottom-up approach, in which the constituting parts of the system to be are analyzed, and agents are identified.

The next chapter details this work, and explain how the agents should behave between each other, as well as towards their respective environment.

5 Design and Implementation of Agents

Following the AMAS theory, we have now to consider all the entities we have described in the domain analysis and, based on the interactions they should have with their respective environment, we are going to make them autonomous, and give them means to actually interact with their environment.

The theory mentions that when the activity of the agents between them, or between them and the environment of the system, are beneficial for each other, then the system is functionally adequate; in other words, it does what it has been designed to do. This state is ideal, and it means that no particular difficulties are encountered by the agents. The behavior in such circumstances is qualified as a *nominal* behavior.

There are, however, circumstances in which the activity of the agents are antinomic with either the activity of other agents in the system, or with the environment of the system. In these cases, the system is no longer functionally adequate, and therefore starts to behave in an undesired way. From an agent point of view, these situations are referred to as Non-Cooperative Situations (NCS). Agents must therefore seek to avoid them, or, if they are in these kinds of situations, seek to escape them.

In this chapter, we first identify all the agents we need to design, based on the previously analyzed entities, and describe their knowledge and abilities. Once all the agents are described, we present their nominal behaviors, and finally, we expose the potential non-cooperative situations, along with means to get back to cooperative states.

5.1 Identification of Agents

In order to have a comprehensive vision of the system, we start by describing all types of agents before detailing their behaviors.

5.1.1 Parameter-Agents

The first type of agent is created after the analysis of the information transmitted between the game adaptation system and the game platform. We have described this information exchange as a set of parameters. **To give autonomy to the parameters**, in order to have them

make decisions, and take the appropriate value depending on their environment, we define an autonomous agent that is named *parameter-agent*. For a scenario S, there is a parameter-agent for each parameter of the set of input parameters $IN(S)$. A parameter-agent is the representation of a parameter within the multi-agent system.

Since parameters define the game experience, and since this experience will eventually need to be adapted, the main ability that is conferred to a parameter-agent consists in the modification of the value of its parameter it represents. This decision should have an impact on the actual game platform as soon as possible, and therefore should influence the game experience.

A parameter-agent is of course aware of the current value of the parameter it represents, along with the name of the parameter given by designers. Moreover, this agent knows the bounds $[p_{i_{min}}, p_{i_{max}}]$ defining the domain definition of the parameter it is associated with.

A parameter-agent has the ability to communicate with its environment. It does not spontaneously try to contact other agents in the system, but it can, however, receive requests from other agents in its environment. The nature of these messages is detailed in the next sections (see 5.2).

5.1.2 Measure-Agents

The second type of agent is created following the analysis of the interactions between the game platform and the game adaptation system. We saw in section 4.2.2 that the information about players' activities on the game platform are sent to the game adaptation system in the form of measures.

These measures are also given a degree of autonomy and pro-activity, by the introduction of *measure-agents*. Similarly to a parameter-agent, a measure-agent is responsible of the value of one measure of the game activity. There is one measure-agent for each measure defined by human experts, this measure-agent is identified by the name given to the measure.

The main knowledge of a measure-agent is the current value of the measure it represents. The activity of the player and the game platform are likely to make the measure value fluctuate over time. Each time this value is modified, the corresponding measure-agent is informed. Additionally this agent knows the bounds that the value has, in other words, all the values the measure can take.

Similarly to parameter-agents, measure-agents have the ability to receive messages from other agents in their environment. They have the ability to send messages to any agent they are aware of. The behavior section details the nature and the purpose of these messages (see section 5.2). A measure-agent responsible for a measure m is aware of other agents present in its environment. These agents are those who are likely to influence the value of the measure m. These influences are listed by the game designers in a correlation matrix which is now described.

Neighborhood Knowledge: Correlation Matrix

To drive the game adaptation mechanism, we need to inject information coming from the knowledge of the domain experts. However, and as we already mentioned in previous sections, we should not expect a comprehensive and consistent model of the game dynamics.

To efficiently and simply **gather information from experts**, we require them to express correlations that may exist between measures on the player activity and inputs of the game. Game designers or pedagogical experts are indeed likely to know which parameters influence which measures in which direction.

Difficulties may arise when it comes to determine the degree of influence of each parameter on a measure, as it may strongly depend on each individual player, on the conditions of the game environment as well as on the progression of both the game storyline and the accomplishment of the player.

For that reason, experts are asked to l**ist correlations that *may* exist** between a parameter and a measure, or between two measures, given that a correlation is defined as follows: a correlation between an input I and a measure M exists if a variation of the value of I is likely to trigger a variation of the value of M. The correlation is said positive if an *increase* (respectively a *decrease*) of I is likely to trigger an *increase* (respectively a *decrease*) of the value of M, and it is said negative otherwise.

When nothing is expressed between two measures or parameters, it simply means that the designer does not think of a direct correlations between these two elements. It may mean that there is no correlation, or that the correlation is not obvious, and consequently, we will not be using it.

To clarify, let us give intuitive but still abstract examples of such relations: a positive correlation may exist between a parameter defining the effectiveness of a resource a player has at hand, and the achievement degree of a task that needs that particular resource to be accomplished. By contrast, a negative correlation may exist between the skill level of an opponent of the player, and the measured success of the player after a confrontation with this opponent.

From these examples, we can understand that these correlations may not remain true at all time. A player can potentially master the game enough so that a modification of these parameters does not have influence on the measured activity. Different players may be differently sensitive to various aspects of the game, making the importance of the required modification to parameters different. This is why only the nature of the correlation (positive or negative) is required from the experts. The impact of this decision will be evaluated in chapter 6.

The provided definition only considers monotonic correlations even though, depending on the game, there may exist non-monotonic correlations between specific inputs and measures. In such cases, we choose to ignore them and solely focus on those that are considered monotonic by human experts.

From experts' knowledge, a matrix is established, listing all the identified correlations between inputs and measures, or between measures and other measures, by their sign: ei-

ther $+$ or $-$. Table 5.1 presents an example of such correlations with two measures ("Measure 1" and "Measure 2"), and four inputs ("input 1" to "input 4"). Five correlations are listed in the table.

- The value of "Measure 1" is
  - positively influenced by
    - Measure 2
    - input1
    - input2
- The value of "Measure 2"
  - negatively influenced by
    - input 2
    - input 3
  - positively influenced by
    - input 4

In the table, some cells are empty. This means that the experts did not mention any correlation between the value of "Measure 1" and "input 3" and "ìnput 4". Similarly, they did not mention any correlation between the value of "Measure 2" and "input 1" and "input 2".

Table 5.1 — Example of correlation matrix. Each measure is correlated to two inputs. Measure 1 is correlated to Measure 2

	Measure 1	Measure 2
Measure 1		
Measure 2	+	
input1	+	
input 2	+	-
intput 3		-
input 4		+

Each measure-agent knows by which agent it is influenced, as well as the sign of the influence (i.e. either positive or negative).

5.1.3 Satisfaction-Agents

Criticality functions are the representations of human experts' requirements and pedagogical objectives. Since there may be a large amount of different requirements, each of them needing different actions to be taken in the game platform. In order to interact locally with other parts of the system, we create *satisfaction-agents* to represent them. Since a satisfaction-agent expresses either a constraint on a parameter or an objective on a measure, and since

both parameters and measures are represented by either a parameter-agent or a measure-agent, the satisfaction-agent is linked to one of those, that we refer to as a *relative* agent. Therefore **a satisfaction-agent expresses either an objective or a constraint on the value of its *relative* agent**. For this reason, each satisfaction-agent perceives at all time the value of its relative agent.

The satisfaction-agent is given all the parameters of the criticality function that the designer expressed: inf, η_1, η_2, ϵ_1, ϵ_2 and sup (see section 4.2.3 for a description of the purpose of these parameters). If some of them are not mentioned, then they are replaced by default values.

Given the value of the relative agent, the satisfaction-agent has the ability to use the parameters of the satisfaction function to determine the satisfaction level of the objective or of the constraint the relative agent represents. The satisfaction value can be computed as many times as necessary; typically, each time the value of the relative agent changes.

Given the value of the relative agent and the parameters of the satisfaction function, the satisfaction-agent may observe that the satisfaction value is not maximal. In which case, it is possible for the satisfaction-agent to determine if the value of the relative agent needs to be greater or lower. This information is required for the nominal behavior of the agent (see section 5.2.1).

Satisfaction-agents can send requests to agents they are aware of. The only agent they are aware of is their relative agent, therefore they can send it requests to modify its value. See section 5.2 for details on what these requests consist of.

5.1.4 Agents Overview

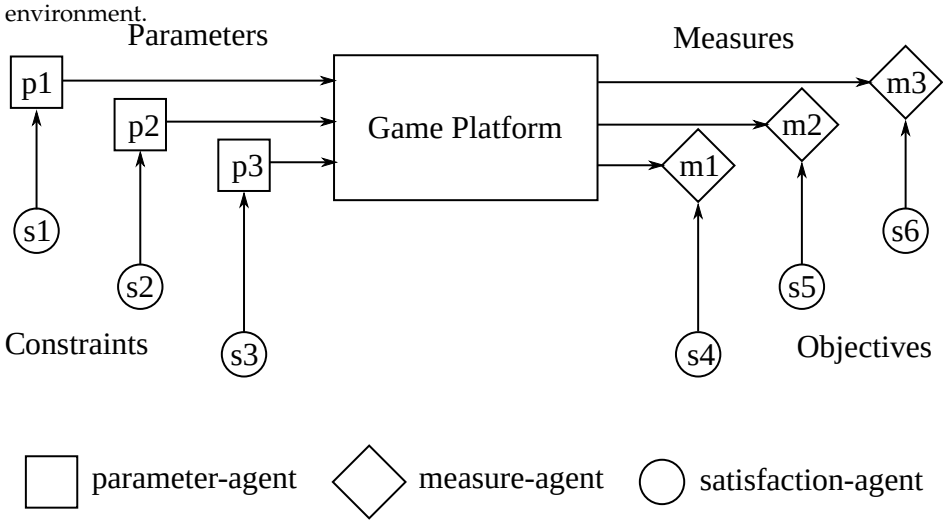
These three types of agents describe all the agents needed in the game adaptation system. They represent all the interactions the game adaptation system has with its environment, and at the same time, the agents represent all the components of the game adaptation system itself. Figure 5.1 represents the game platform (and implicitly the player who interacts with it) in interaction with the game adaptation system, represented here only by instances of the three types of agents.

The next section describes the behaviors of the agents, as well as their interactions.

5.2 Nominal Behaviors of Agents

The key aspect of the game adaptation system is the behavior of the various agents, since it is how these agents handle their interactions that determines its global function of the system. Therefore care must be taken while designing their goal and decision process. This section describes the various **behaviors in the nominal case**, that is, their standard behaviors, **where no cooperation conflict occurs between agents**, or between them and the environment of the system.

Figure 5.1 — Instances of the three types of agents in interaction with their respective environment.



5.2.1 Satisfaction-Agents

The activity of the system is initiated by satisfaction-agents. Whether a satisfaction-agent represents an objective on a measure or a constraint on an input parameter does not make a difference. **The agent always has a local goal: to satisfy the objective or constraint it represents**. In order to be satisfied, it needs to compute a minimal criticality value, which means that the value of its relative agent needs to be in the optimal interval of its criticality function. The behavior of a satisfaction-agent is described by the algorithm 5.1.

Algorithm 5.1 — Satisfaction-agent behavior

```
1 while true do
2     relativeValue ← getRelativeAgentValue();
3     criticality ← computeCriticality(relativeValue);
4     if criticality > 0 then
5         sign ← getRequiredVariationSign(relativeValue);
6         request ← createRequest(criticality, sign);
7         sendRequest(request, relativeAgent);
8     end
9 end
```

The first step of the behavior of the satisfaction-agent is to observe the value of its relative agent (line 2). Once the value is known, the agent computes its criticality, by using the criticality function defined in 4.2.3 (line 3). If the value of the relative agent is optimal, then the computed criticality is null. If the value of the relative agent is not optimal, then the criticality is in the interval $[0, 100]$.

In this latter case, the objective of the agent is to have the value of its relative agent to be modified so that the computed criticality is lowered (line 4). Since that value cannot be

modified directly, the only option of the satisfaction-agent is to send a message to its relative agent in order to ask a modification of the value.

The message sent to the relative agent is rather simple. It contains only two pieces of information. First, there is the criticality value computed by the objective agent, which is a value comprised between 0 (not included) and 100. This value describes the degree of "non-satisfaction" of the sender. Then, the satisfaction-agent includes the desired variation of the relative agent. As described in section 5.1.3, the satisfaction-agent is able to determine if its relative agent needs to raise or to lower its value in order for the computed criticality to go down. The requests therefore contain a criticality value as well as a desired variation sign (either *up* or *down*) (line 7).

5.2.2 Measure-Agents

As described in section 5.1.2, a measure-agent is responsible of a measure on the game activity. Therefore at all time, **such an agent tracks the evolution of the value provided by the game platform**. In itself, the agent does not have specific requirements other than that. But being a cooperative agent, **it constantly checks if there are requests addressed to him.** Its behavior is described by the algorithm 5.2.

Algorithm 5.2 — Satisfaction-Agent Behavior

```
1  while true do
2      updateObservedValue();
3      receivedReq ← receiveRequest();
4      if receivedReq ≠ null then
5          requestedSign ← receivedReq.sign;
6          criticality ← receivedReq.criticality;
7          foreach correlation in correlationList do
8              newSign ← requestedSign;
9              if correlation.isNegative() then
10                 newSign.reverse();
11             end
12             newRequest ← createRequest(criticality, newSign);
13             sendRequest(newRequest, correlation.agent);
14         end
15     end
16 end
```

When no messages are received, the agent proceeds to the next step, where it simply updates its value again (line 2), and checks for messages (line 3). If a message is received, the agent needs to process it. The received message is always of the same type. It contains a criticality value, and a direction in which its value needs to be modified.

Since a measure-agent seeks to **maintain cooperative interactions with other agents** present in its environment (in order to not have an antinomic activity with them), when such a message is received (line 4), it tries to help the agent that sent that request. However, the measure-agent does not have the ability to modify the value it represents, as this value is obtained by observing the activity on the game platform. Consequently, in order to modify this value, **the measure-agent seeks help from the other agents it is aware of**, that is the parameter-agents it is correlated to (line 7).

The measure-agent **propagates the requests it has received in the first place to these parameter-agents**. However, these propagated requests may need change before being sent. Indeed, if the correlation between the measure-agent and a parameter-agent is negative, then the sign of the needed variation in the request needs to be inverted. If it is not inverted, the request may be counter-productive (line 9).

5.2.3 Parameter-Agents

A parameter-agent is responsible of the value a of single parameter. As described in section 5.1.1, it has not only **the ability to modify the value of the parameter** it is responsible for, but it has also communication skills, and more precisely, **it is able to receive and process requests**.

5.2.3.1 Behavior

Algorithm 5.3 describes the behavior of the parameter-agent. On its own, a parameter-agent has no specific activity. Therefore it awaits for requests coming from other agents. Requests it receives can be initiated by either satisfaction-agents that represent constraints on the parameter value defined by the human expert, or by measure-agents that are correlated with this parameter. Either way, the parameter-agent does not need to make the distinction, and processes requests indifferently (line 2).

Algorithm 5.3 — Parameter-Agent Behavior

```
1 while true do
2     receivedReq ← receiveRequest();
3     if receivedReq ≠ null then
4         requestedSign ← receivedReq.sign;
5         AVT.updateParameterValue(requestSign);
6     end
7 end
```

Since a parameter-agent is cooperative, each time it receives a request from another agent, **it tries to help the sender by modifying the value of the parameter**. Since the request only contains the sign of the variation, and not the optimal value that the parameter should take, it is likely that a single modification of the parameter value will not be enough to satisfy the sender of the request. Large modifications of the value could accelerate the

process, but they lack precision when it comes to find an accurate value. For that reason, **the parameter-agent adopts a strategy to modify the value that is simultaneously fast and precise** (line 5), that is, the *Adaptive Value Tracker* (AVT) that is now detailed.

The goal of an Adaptive Value Tracker (AVT) is to track a dynamic value in a given range, according to successive inputs of the type *greater* or *lower* [Lemouzy et al., 2011].

Each time two successive inputs are identical, the modification speed of the AVT is increased. Each time two successive inputs are opposite, it is decreased to gain accuracy. More details about how the AVT behaves can be found in the appendix B.

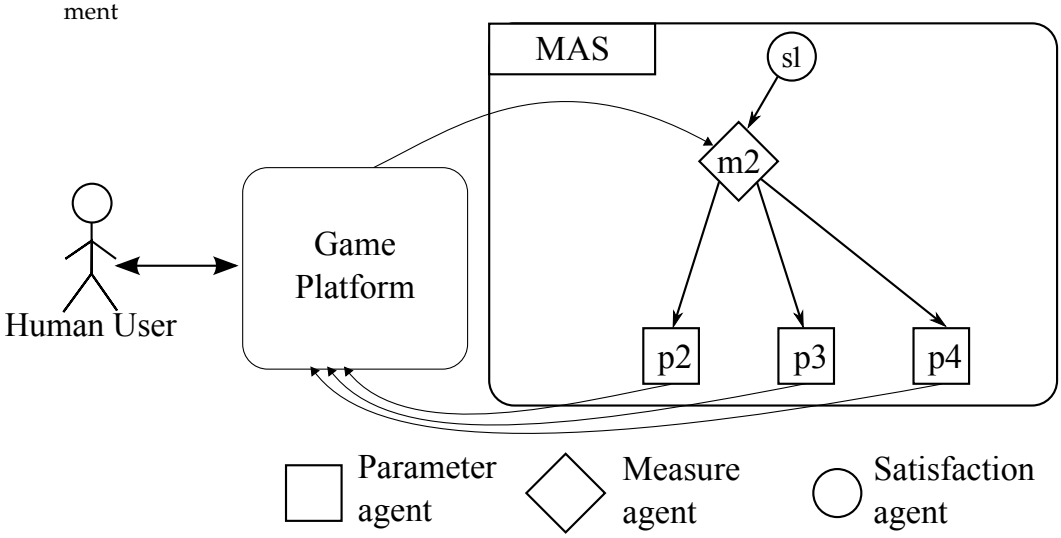
This strategy allows to **quickly move through the search space when the optimal value is far away** from the current value, and at the same time, **it allows for precise modification when the optimal value is close** to the current value of the AVT.

5.2.4 Multi-Agent System Activity Overview

To summarize, the information flow in the system is rather simple. The activity of the player on the game platform generates data that go in the system through the measure-agents. Requirements of human experts represented by satisfaction-agents interact with these observations and generate requests. These requests use knowledge from experts to navigate through the agent graph and generate activity that adapt the challenge to players and make the game behave within the boundaries defined by game designers.

Figure 5.2 shows the typical interaction flow between the game and the agents. The game platform updates a measured value, the satisfaction-agent observes the modification and sends a requests to the agent responsible for the measure, which then sends requests to relevant parameter-agents. Parameters values are sent back to the game platform, which hopefully makes the initial measured value closer to its objective.

Figure 5.2 — Overview of the agents interactions with each other and with the environment



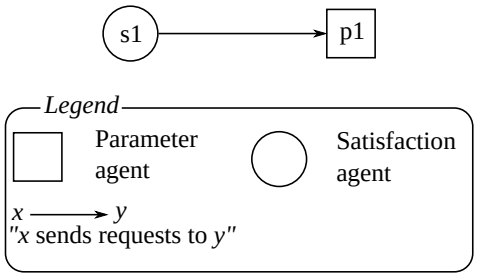
In order to illustrate the behavior of the multi-agent system as we have described it simplest problem, to more complicated test cases to explore various situations in which the

multi-agent system may be in.

5.2.4.1 Test Case 1: Single Parameter with a Single Constraint

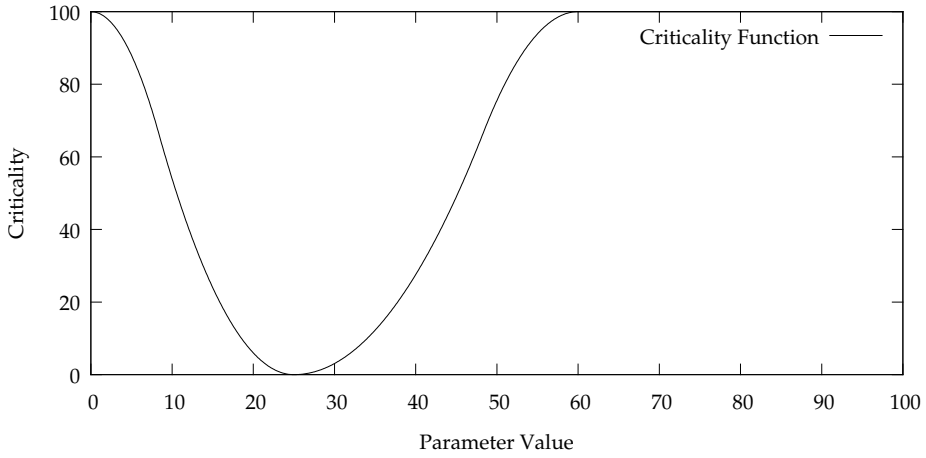
The simplest test case that can be imagined is constituted by a single parameter. In order to observe the activity in the multi-agent system, we still need to introduce a requirement on that parameter. Consequently, the simplest multi-agent system we can instantiate is composed of a parameter-agent and a satisfaction-agent that expresses a criticality function on the parameter value (see figure 5.3).

Figure 5.3 — Test case 1: 1 parameter and 1 constraint



In the following example, we consider the parameter-agent *p1*, and arbitrarily define its range as the $[0, 100]$ interval. The criticality function is defined so it is centered on the 25 value (see figure 5.4).

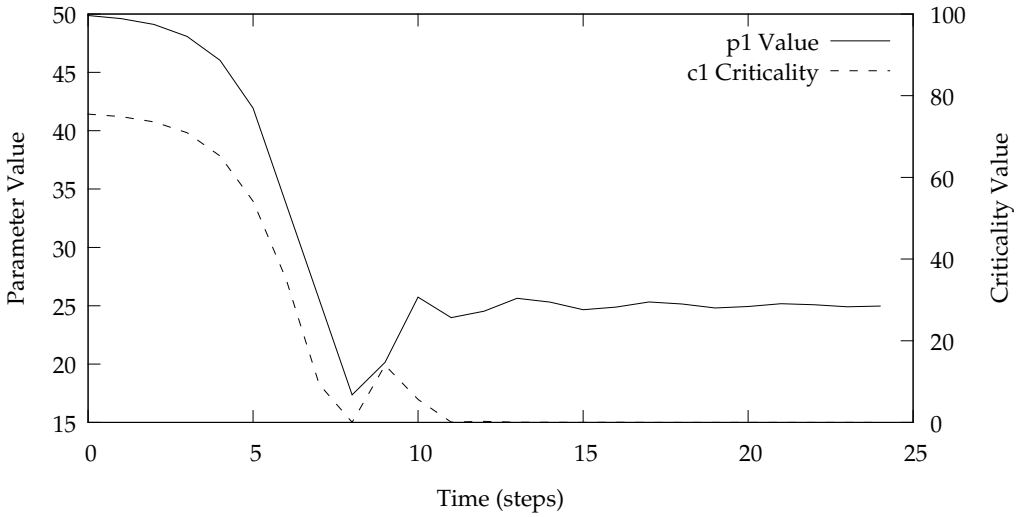
Figure 5.4 — Criticality function for the satisfaction-agent in the test case 1



The system is instantiated with this simple problem, and then ran for a few steps. At each step, an iteration of the agent behavior (as described in section 5.2) is executed. Figure 5.5 shows the execution log.

What we observe is the typical Adaptive Value Tracker (AVT) behavior (see section 5.2.3). Requests are sent from the satisfaction-agent to the parameter-agent, which listens to them and changes its value accordingly. At the beginning of the process, the parameter is far from the optimal value, and therefore receives several identical consecutive requests. Con-

Figure 5.5 — Test case 1: Execution log



sequently, until step 8, the variation of the parameter is large, and at step 8, it goes beyond the optimal value. In parallel, the dashed line shows that the criticality quickly lowers, and slightly raises when the parameter goes beyond the optimal value.

The satisfaction-agent then sends an opposite request, and the parameter-agent changes its variation sign, and reduces its variation speed. It then oscillates closely to the optimal value and finally converges around step 20. The criticality is rapidly negligible after step 11.

5.2.4.2 Test Case 2: Single Parameter with Single Varying Constraint

The next test case we propose to run is the case where the constraint relative to a single parameter is varying over time. In the previous test case, we have shown that a simple constraint on a parameter can be rapidly satisfied. However, since the problems we address are dynamic by nature, it is likely that the optimal value that a parameter should take is not constant, and changes over time.

Consequently, we propose to take the same test case as before, with the same agents (see figure 5.3), and the same criticality function (see figure 5.4), except that after some time, we change it so the optimal value is no longer the same (see figure 5.6). The execution log of this test case is shown in figure 5.7.

We observe the same behavior as in the previous test case, the value of the parameter is stabilized after step 11, and the criticality is null. The system is stable, until step 20, where we arbitrarily change the satisfaction function of the satisfaction-agent $c1$, by shifting it so the optimal value is now 51. As a consequence, at step 21, the criticality is suddenly higher, showing the newly appeared non-satisfaction. This criticality value generates activity among the agents, and the value of the parameter starts to increase. Around step 30, the variation of the parameter value becomes significant, and at step 35, it goes beyond the optimal value. Since the slope of the criticality function is much stiffer around these values, the criticality goes as high as 85 when the parameter value is too high.

Figure 5.6 — Test case 2: Second criticality function

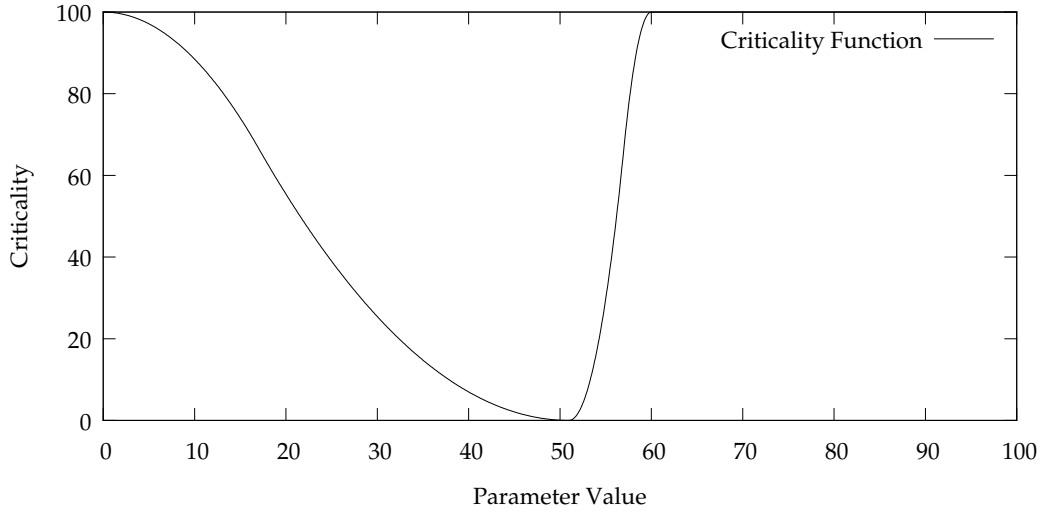
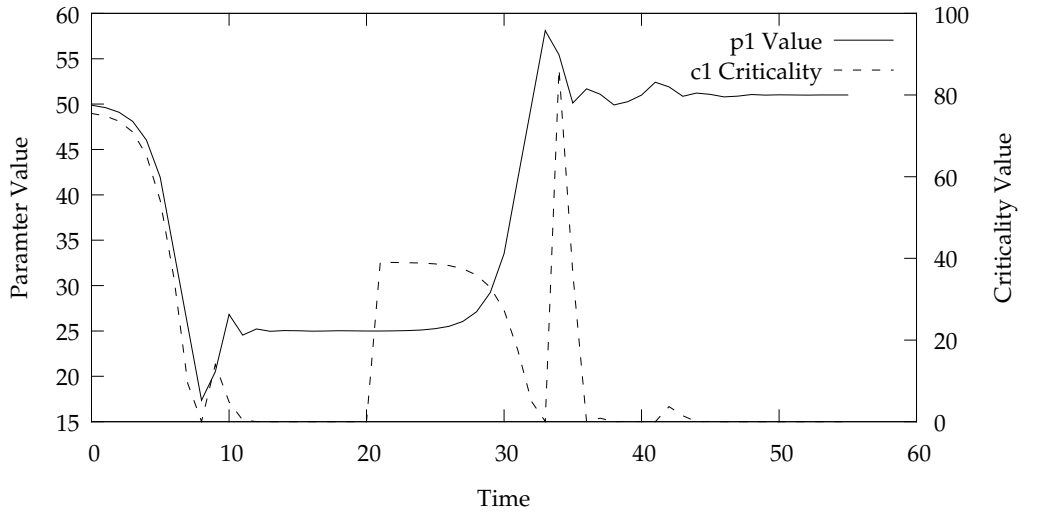


Figure 5.7 — Test case 2: Execution log



Quickly, the same dynamics as before is observed: the value oscillates around the optimal, the criticality is rapidly negligible, and the system is stable around step 46.

We observe in this simple test case that the convergence on the first objective has no significant consequences on the behavior of the system when an objective changes. The agents are able to converge as rapidly as if it was their first objective.

In the following test cases, we introduce the third type of agent: measure-agents. Since our objective is only to exhibit simple representative situations in which the system may be into, we design simple means to obtain measured values.

5.2.4.3 Test Case 3: Mock Measure with an Objective

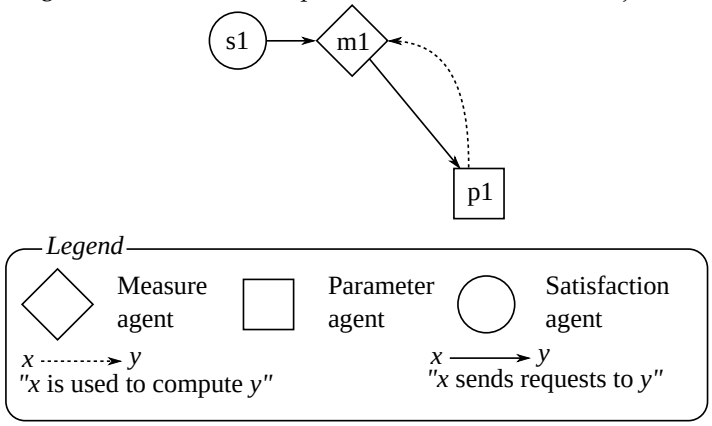
We now propose to exhibit situations in which measure-agents are involved. Measure-agents are given values that are observed in the game. Typically, such values are the result

of the activity of the player's interactions with the game with specific values for its input parameters. In the simple situations we expose here, we propose to build simple models to give a value to the measure-agents. This, by no means, simulates the complexity of real-world applications, but it enables us to observe if the mechanisms we have described so far are able to deal with specific situations.

The first test case we design that involves measure-agents is really simple, and quite similar to the first test case we described (see 5.2.4.1). We first create a parameter-agent $p1$ that can take any value in the $[0, 100]$ interval. In addition, we create a measure-agent $m1$. The simplest way to give a value to the measure-agent is for it to take the value of the parameter.

Then, we create an objective on that measure, by introducing a satisfaction-agent $s1$ relative to $m1$, with the same criticality function as defined in the first test case (see figure 5.4). The satisfaction-agent, in order to modify its value, sends requests to the parameter-agent. These agents and their interactions are depicted on figure 5.8.

Figure 5.8 — Test case 3: 1 parameter, 1 measure and 1 objective



The execution of this test case is the same as the first one. The value of $m1$ equals the value of $p1$, and therefore the same requests are sent to $p1$ as in the first test case, except that these requests now transit through $m1$. The execution log of this test case is not shown, since the figure would be the same as figure 5.5, with the value of $m1$ strictly equal to the value of $p1$.

Now that this type of mock measure-agent has been introduced, it is used in the next test case in order to show more interesting behaviors of the multi-agent system.

The system, as it has been described so far, can be used to find optimal parameter values to have the player satisfy all objectives, and parameters satisfy all constraints. All the interactions between agents we have described are cooperative, and the activity of agents are beneficial for their environment.

However, there are situations in which the system cannot function as it is now. Since the global function of the system emerges from the interactions of the agents, problematic situations may lead the system to no longer be functionally adequate.

In the next sections, we describe situations in which interactions between the agents may become antinomic. These situations are analyzed, and mechanisms are proposed to let the

agents get back in states in which they have cooperative interactions with their environment.

5.3 Concurrency Between Requests

When several agents interact with each other, it is likely that situations occur where a parameter-agent receives **at the same time different requests that are contradictory**. The behavior of this agent, as we have described it so far, simply obeys to all the requests it receives. Therefore, when two contradictory requests are received, a parameter-agent has to take a decision that will be counter-productive for one or several agents that required the parameter-agent's help.

5.3.1 Behavior Improvement

The AMAS theory gives us insights on how to handle these kinds of situations. The involved parameter-agent first needs to be able to evaluate how important are its own actions to the agents that sent the requests. This can be done by **comparing the criticality values embedded in each received request**. The agent should then **help the agent believed to be the most critical**. Helping the most critical agent at the local level is the most cooperative attitude, and will therefore lead the system, at the global level, to have cooperative interactions with its environment, and therefore to be functionally adequate.

The behavior of the parameter-agent is therefore updated to include this decision process. This newer version is depicted in the algorithm 5.4.

Algorithm 5.4 — Parameter-Agent behavior with the request selection mechanism

```
1 while true do
2 |   allRequests ← receiveRequest();
3 |   if allRequests ≠ null then
4 |   |   worstRequest ← findRequestWithHighestCriticality();
5 |   |   requestedSign ← worstRequest.sign;
6 |   |   AVT.updateParameterValue(requestSign);
7 |   end
8 end
```

We now propose to examine a series of test cases where this newly introduced mechanism is involved. This will allow us to effectively observe how these non-cooperative situations are handled by the multi-agent system, and also to determine if other problems may have remained undetected.

5.3.2 Test Case 4: Cooperation Between Parameters

Let us consider the case where we have a measure-agent that depends on two parameters. For the sake of simplicity, we consider that the measure value is the mean of the two param-

eters values. We create a measure-agent $m1$, and two parameter-agents $p1$ and $p2$. Similarly to our previous test cases, we arbitrarily state that $p1$ and $p2$ can take any value in the $[0, 100]$ interval, and the value of $m1$ is $\frac{p1+p2}{2}$. Consequently, to modify its value, $m1$ sends requests to both $p1$ and $p2$.

We introduce two satisfaction-agents: $s1$ expresses an objective on the value of $m1$, and $s2$ expresses a constraint on the value of $p1$ (see figure 5.9). The criticality function of $s1$ is centered around the optimal value of 45, and the criticality function of $s2$ is centered around the optimal value of 35 (see figure 5.10).

Figure 5.9 — Test case 4: 2 parameters, 1 measure and 2 objectives

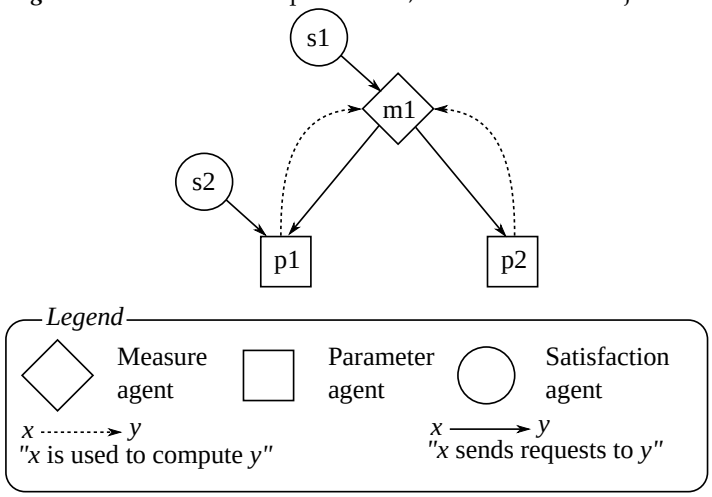
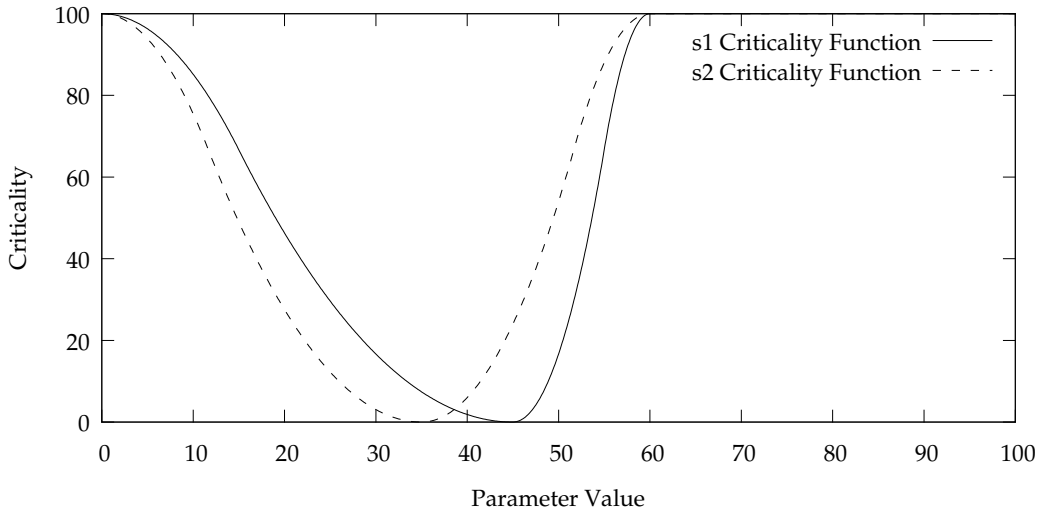


Figure 5.10 — Test case 4: 2 criticality functions



The execution log is shown in both figures 5.11 and 5.12. At the beginning of the execution, the two values of $p1$ and $p2$ are both equal, and consequently, the value of $m1$ is equal to 50 as well. Given the two criticality functions we have expressed, the two criticalities of $s1$ and $s2$, respectively related to the values of $m1$ and $p1$ are both non-null, the criticality of $s1$ being greater than that of $s2$.

Figure 5.11 — Test case 4: parameters and measure values evolution

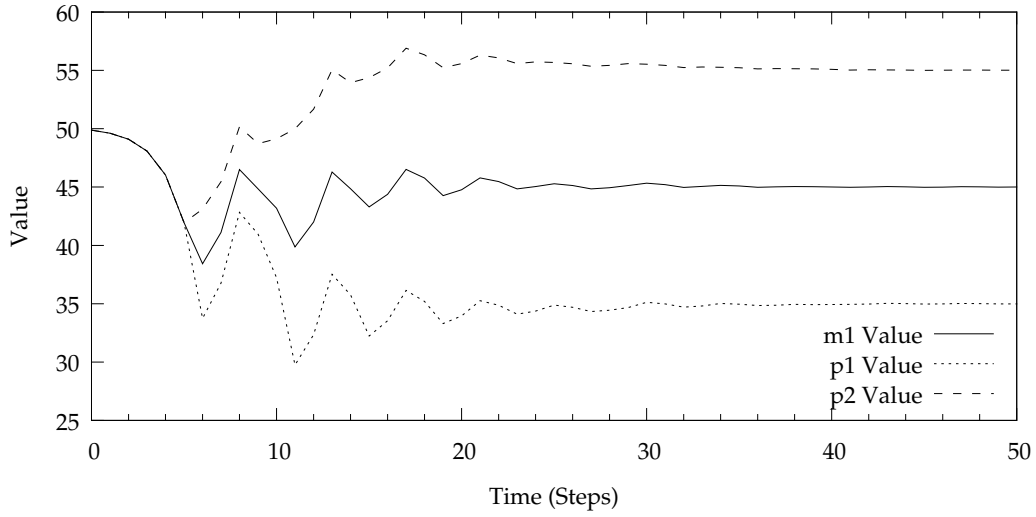
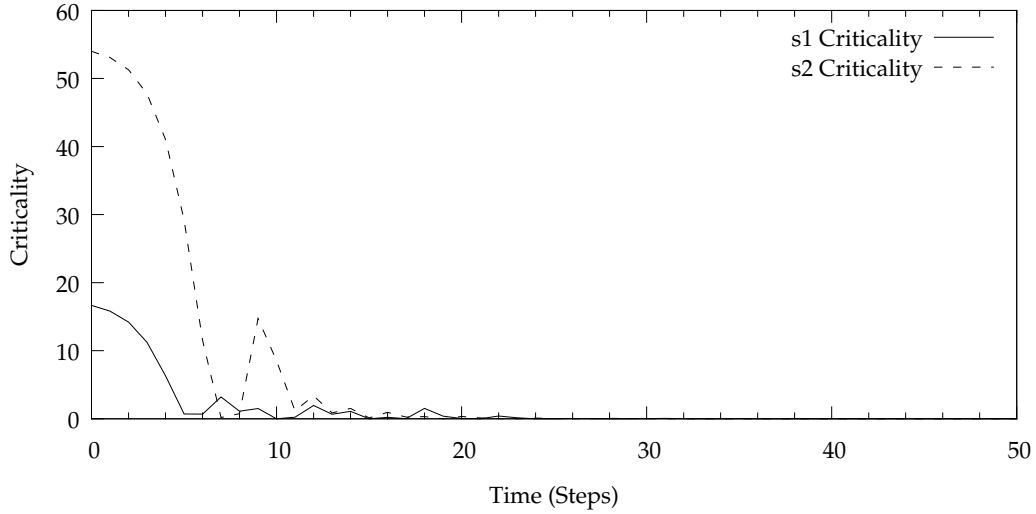


Figure 5.12 — Test case 4: Objectives Satisfactions



Therefore the two satisfaction-agents $s1$ and $s2$ start by sending requests to $m1$ and $p1$ to lower their values. $p2$ receives requests only from $m1$, and lowers its value. $p1$ receives requests from both $m1$ and $p1$. At the beginning, those from $m1$ are more critical, and therefore, they are those $p1$ listens to.

At step 5, we observe that the value of $m1$ goes beyond its objective value, and reaches the value of 42. Consequently, $p2$ is now requested to go up, hence the observed variations after step 5. $p1$ on the other hand, now receives two contradictory requests. Requests from $m1$ demanding its value to go up, and requests from $s2$, still demanding to go down. At that point, $s2$ is still more critical, and therefore, $p1$ goes down.

At step 6, $p1$ goes beyond the optimal value of $s2$. Therefore, all the requests it now receives are demanding it to go up, hence the increase of its value. At step 7, even though $p1$ is greater than 35 again, since now $m1$ is more critical, $p1$ still increases its value. At step 8, the two parameters decrease their value, leading to a state, at step 9, where the two agents

evolve again in two different ways.

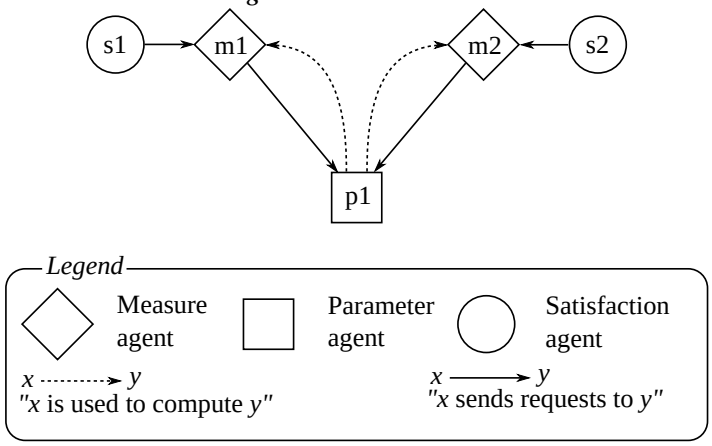
The same dynamics are repeated over the next steps, and the agents change their variation sign, they gain accuracy, thanks to the behavior of the adaptive value tracker. By always helping the least satisfied agent, i.e. the most critical, the system manages to reach a state in which all criticalities are lowered until they are null. We observe that $p2$'s value varies because of the requests sent by $s2$, even though these agents do not communicate. In other words, the satisfaction of $s2$ and $p1$ alter the satisfaction of $s1$, and consequently, $s1$ tries to use the degree of freedom is has ($p2$) to be satisfied again without degrading the situation of $p1$.

It is possible however, that in certain cases, there are not enough latitude for the agents to offer a situation in which all constraints and objectives are satisfied. The next test case is an example of this type of situation.

5.3.3 Test Case 5: Competing Objectives

In this test case, we create only a single parameter-agent $p1$ which, as usual, can take any value in the $[0, 100]$ interval. We then introduce two measures $m1$ and $m2$ that take the same value as $p1$. Finally, two satisfaction-agents $s1$ and $s2$ are introduced. $s1$ is relative to the value of $m1$, and has a satisfaction function with the optimal value being 35, and $s2$ is relative to the value of $m2$ and has a criticality function with the optimal value being 25. The organization of the agents is shown in figure 5.13, and the two criticality functions are shown in figure 5.14. The execution log of this test case is in figure 5.15.

Figure 5.13 — Test case 5



The interesting behavior in this case is, of course, the behavior of $p1$. $m1$ and $m2$ send requests to $p1$ demanding a variation as long as their value, and thus, the value of $p1$, is different from their objectives, respectively 35 and 25.

At the beginning of the execution, $p1$ receives requests from both $m1$ and $m2$ demanding that the value is lowered. Hence, the observed variation in the first 6 steps. At step 6, the value of $p1$ is around 33. This means that the requests sent by $m1$ are now demanding an opposite variation, but since the criticality of $s1$ is lower than that of $s2$, $p1$ continues to lower its value; in other words, it listens to the most critical.

Figure 5.14 — Test case 5: criticality functions

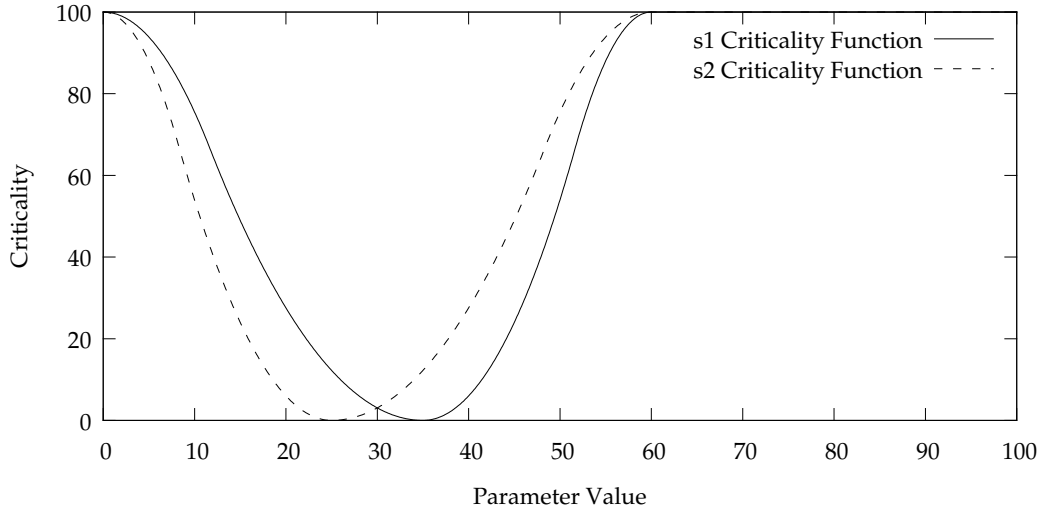
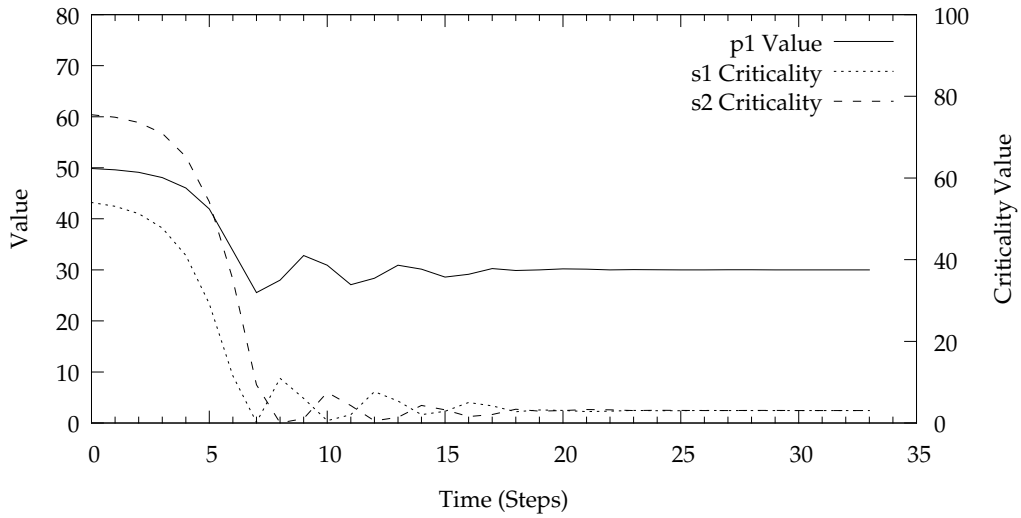


Figure 5.15 — Test case 5: Execution log



By continuing to lower its value, $p1$ makes the criticality of $s2$ lower, but it also makes the criticality of $s1$ greater. Eventually, the criticality of $s1$ becomes greater than the criticality of $s2$ and therefore, $p1$ changes the sign of its variation. This is observed at step 8, where the value of $p1$ is increased.

The same dynamics is observed in the following steps, and $p1$ alternatively increases and decreases its value to satisfy the most critical agent. Again, thanks to the behavior of the adaptive value tracker, more and more accuracy is gained by $p1$, and eventually, $p1$ stabilizes at the value 30 around step 20.

At that moment, both $p1$ and $p2$ have a non null criticality (around 3), and they are not completely satisfied. This is what we consider to be the optimal state, since there are no possibilities to satisfy the two constraints simultaneously.

Another way to look at it is to introduce the notion of *global criticality* to reflect the satis-

faction of the system as a whole. This global criticality is a function defined as the maximum of all the criticality functions defined in the system. Therefore in this test case, this function is the maximum of $s1$ and $s2$ criticality. By looking at figure 5.15, we can observe that the global criticality is at its lower point when the system is stabilized.

We consider then problem of concurrent requests to be solved by the mechanism we just introduced. In the next section, we describe a second type of situation in which interactions between agents may be antinomic, provoking the system to be functionally inadequate. Similarly to the concurrency problem, we propose mechanisms to deal with these situations, and we propose several test cases.

5.4 Delayed and Asynchronous Interactions

In all the situations we have described so far, even if various aspects of the interactions between agents have been investigated, there are still assumptions that were made that need to be analyzed.

Each time an agent takes a decision, and does an action (modifies a value, or sends a request) at a given step, it expects the consequences to happen necessarily in the following step. Similarly, when we proposed to introduce measure-agents along with arbitrary means to produce a value for these measures, these measures were calculated instantly, and as soon as a parameter value was modified, all the involved simulated measured values were updated instantly.

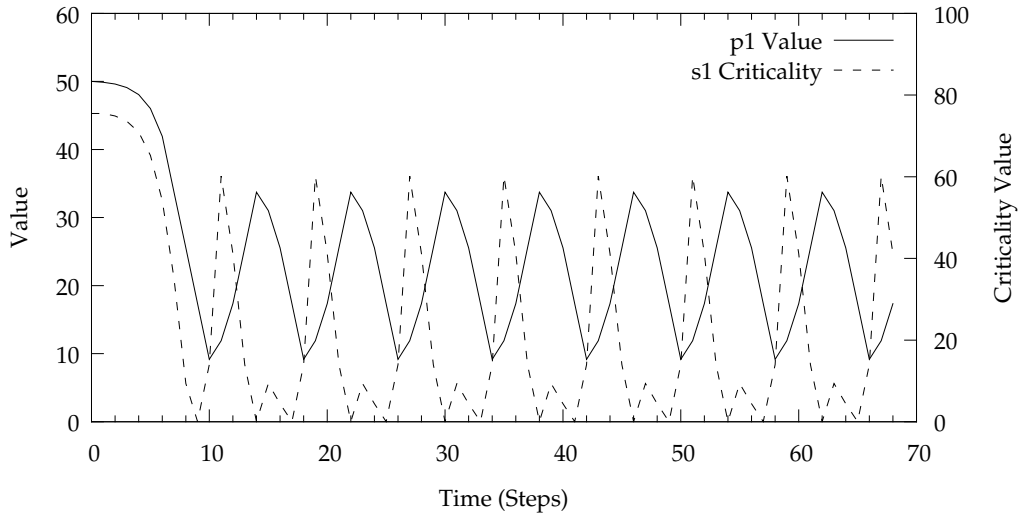
In reality, such an assumption should be considered with care, since there may be many situations in which it would not hold true. First, if we were to handle a large number of parameters, measures, and objectives at the same time, it is unlikely that we would be able to determine the adequate order for all the agents to be executed. Secondly, when a human interacts with a game, even optimal modifications on the parameters of the game may take some time before the optimality is reached, since consequences on the game or on the player may be delayed by various time steps, depending on their nature or on their contexts.

As an example, let us consider the first test case we described (see section 5.2.4.1), with only one parameter-agent $p1$, and a single satisfaction-agent $s1$, satisfied when $p1$ equals 25. We remove the assumptions on the order, by executing all the agents at the same time. The two agents perceive their environment and then, these two agents, in a random order, either send a request or modify their value if they can. This sixth test case execution log is shown on figure 5.16.

We observe that the system does not reach an equilibrium and is not even able to find this trivial solution. At the beginning of the execution, $s1$ sends requests to $p1$ to lower its value. $p1$'s value is decreased, until it goes beyond the optimal value of 25. Then we observe that the value of $p1$ oscillates around 25, without reducing the amplitude of the oscillation, and thus, without stabilizing on the optimal value.

What happens after step 10 is as follows. In step 12 for instance, $s1$ observes $p1$'s value, and determines that this value needs to be increased. Therefore, it sends $p1$ a request, demanding it to increase its value. In the next step (step 13), $p1$ receives the request. In parallel,

Figure 5.16 — Test case 6: Test case 1 without assumptions on the execution order of the agents



$s1$ observes $p1$, that has still not done anything. $s1$ observes that $p1$ is still too low, and therefore sends a new request. Still in the step 13, $p1$ increases its value, and is now equal to 25, which is optimal. Except that $s1$ already sent its request, hence the unwanted modification of $p1$ at step 14. This dynamics is repeated in the following steps provoking these oscillations.

This can be generalized by saying that $s1$ **should have waited for the consequences of the requests it sent before sending new requests**. Therefore, we propose that agents feature **their own mechanism to determine how long they need to wait to consider that consequences of their actions are effective**.

5.4.1 Behavior Improvement

In order to prevent this kind of oscillations around an optimal value or, more generally, to prevent agents from acting before the consequences of their previous actions are effective, **we propose a mechanism that let the agents estimate the delay between their actions and their consequences**. By having knowledge on that delay, agents can anticipate it, and prevent the oscillations observed in the previous test case.

The idea is that a parameter-agent $p1$ that receives requests from a measure-agent $m1$ should estimate a δ_{m1} value that represents the number of steps needed before the consequences of the actions of $p1$ on $m1$ are effectively observable by $p1$. To estimate this value, $p1$ needs to analyze the requests it receives. In the beginning, we propose that $p1$ acts the same way we have described in the previous test case, and thus, begins the oscillations pattern we have seen.

We introduce the concept of *phase*. For a parameter, a *phase* is a series of consecutive steps, in which a given agent sends requests with the same variation sign. Oscillations, as seen in the previous test case, thus consist of a succession of opposed phases.

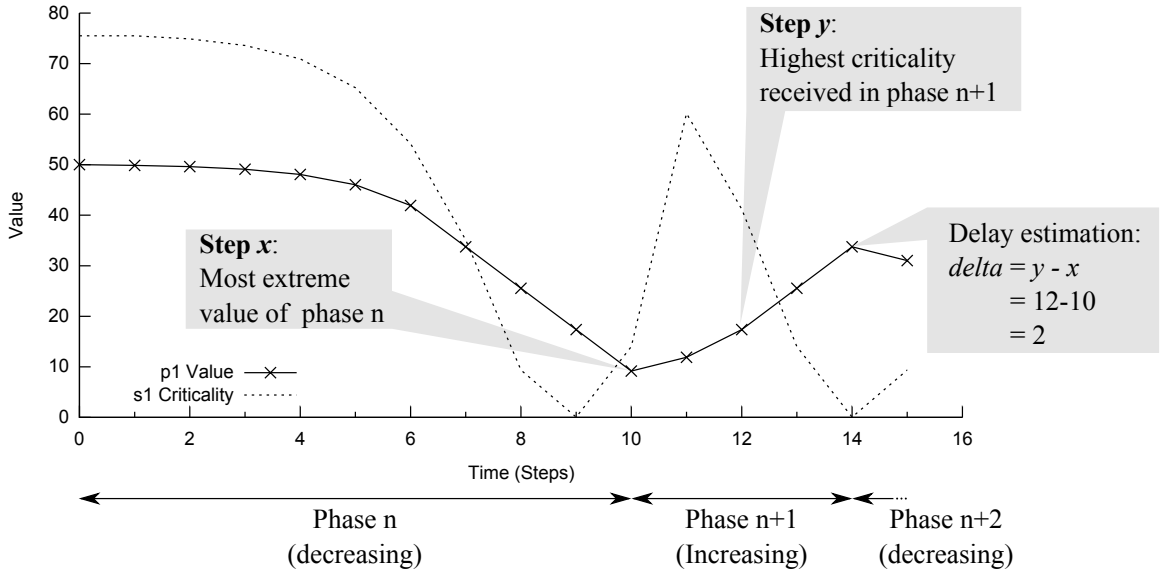
To estimate the value δ_{m1} of the delay of $m1$, the parameter $p1$ uses this concept of phase.

Figure 5.17 shows the beginning of the execution of the test case seen in figure 5.16. Values are exactly the same, the only difference is that the chart is truncated. In a decreasing phase n, requests demanding to lower the value are received. $p1$ decreases the value several times, goes beyond the optimal, and at some point in time, finally receives a request demanding to increase its value. During this decreasing phase n, the step in which the lowest value has been reached is memorized as x.

$p1$ now increases its value, and even though the value is increased, the criticality in the requests coming from $m1$ is still increased. During that increasing phase $n+1$, the step at which the highest criticality is received is memorized as y. Note that the step at which $m1$ is the most critical and the step when the highest criticality is received by $p1$ are not the same, due to communication delay. When the phase $n+1$ is over, the delay can be estimated.

We suggest that the request sent with the highest criticality in the phase $n+1$ is a consequence of the most extreme value of the phase n. Therefore, we can estimate the number of steps it takes for a consequence of an action to be effectively observable by calculating the difference between the number of these two steps. In this case, $y - x = 12 - 10 = 2$. In other words, we observe that the consequences of actions are observed with a two-step delay.

Figure 5.17 — Estimation of the delay δ_{m1} between the actions of $p1$ and the effectiveness of the consequences on sl



Once the delay is estimated, the behavior still needs to be updated. In order to not oscillate around optimal values, **agents need to wait for the consequences of their previous actions before they act again**.

In order to formalize the decision process, we introduce a set of object-oriented structures. These structures will help the agents to properly reason about their environment, and to take the right decisions. The first set of structures, shown in figure 5.18, defines the representations that a parameter-agent has about other agents that sent him requests. Each parameter-agent has a *Representations* attribute, which consists of an instance of *ClientRepresentation* for each agent that sends a request. Each instance of *ClientRepresentation* first

contains the identifier *id* of the involved client, and the representation of three *phases*, as we have introduced them at the beginning of the section: the current, unfinished phase, and the two last, terminated phases, here named *previousRepresentation* for the previous phase, and *previousPreviousRepresentation* for the one before.

Each *Phase* contains the requested variation sign of that phase (*way*) and the representation of all the steps that compose that specific phase. Each step is represented by a *StepRepresentation* object, that contains the step number *stepNB*, the current criticality of the request sender *senderCriticality*, and the current value *myValue* of the agent that owns these representations.

Thanks to these representations, the delay of a particular request sender can be easily obtained, by calculating the difference between the step number of the most extreme value of the *previousPreviousRepresentation* and that of the highest criticality of the *previousPhase*.

Figure 5.18 — Conceptual artifacts to describe agent representations in the algorithmic description of the decision process

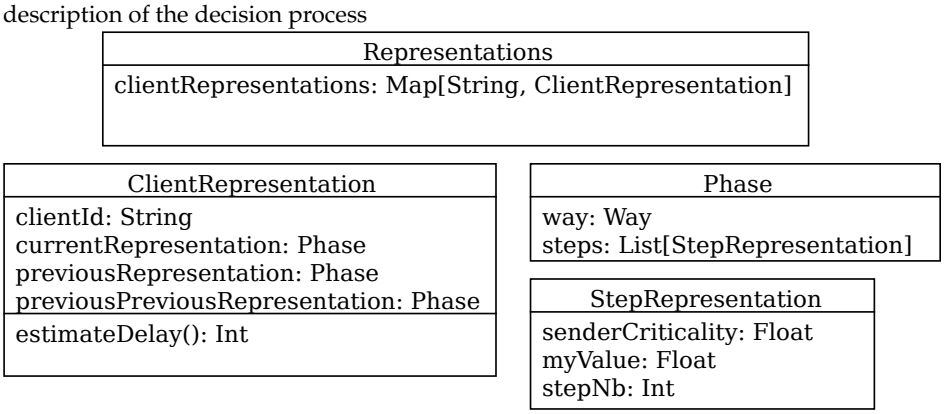


Figure 5.19 introduces the concept of *actions* that are taken by agents. An *action* is used by the parameter-agents to express the fact that they modify their value, and they now need to wait before the consequences are effective. Thus, requests demanding the same variations should be ignored while an agent waits for the consequences to happen. When an agent does an action, it needs to memorize what agent it was doing the action for (*beneficiaryId*), in which direction it modifies its own value (*way*, being either *up* or *down*), what is the supposed delay before consequences can be observed (*delay*). The agent should also count the number of steps it still has to wait before the consequences of the value modification should be effective (*remainingStepsToWait*). The boolean value *actionExecuted* is only introduced to ease the expression of the algorithm.

Algorithm 5.5 describes the updated behavior of the parameter-agents. A parameter-agent starts by receiving all the requests (line 2). Then, it updates all the representations is has about the agents that sent requests (line 4).

Once the representations are updated, the request with the highest criticality is selected (line 5). If no action has been started, or if a previously action is now over (line 7), a new action is created, based on the worst request received (line 8).

Then comes the action part of the algorithm. If a current action is defined, and not over (line 10), then the action should effectively be executed (line 13) if it has not been executed

Figure 5.19 — Conceptual Artifacts to describe the representation of a parameter-agent actions in the algorithmic description of their decision process

Action
beneficiaryId: String way: Way delay: Int actionExecuted: Boolean remainingStepsToWait: Int
«Constructor» Action(beneficiaryId: String way: Way delay: Int)

yet (test line 11), otherwise the number of steps to wait should just be decreased (line 16).

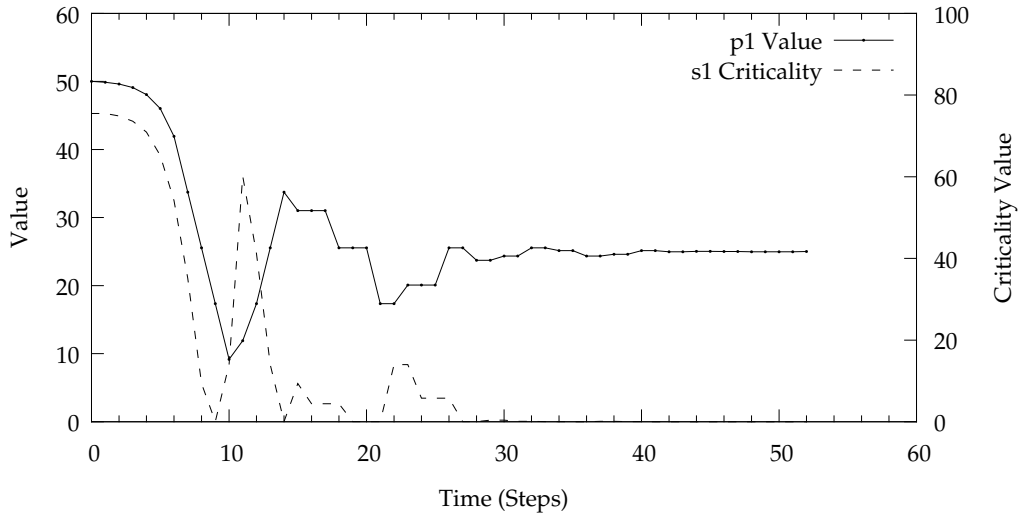
Algorithm 5.5 — Parameter-Agent behavior with the request selection mechanism

```
1  while true do
2      allRequests ← receiveRequest();
3      if allRequests ≠ null then
4          representations.update(allRequests);
5          worstRequest ← findRequestWithHighestCriticality();
6          ClientRepresentation client ← representations.get(worstRequest.senderId);
7          if currentAction == null || currentAction.isOver() then
8              currentAction ← new Action(worstRequest.senderId, worstRequest.way,
               client.estimateDelay());
9          end
10         if currentAction ≠ null ∧ ¬ currentAction.isOver() then
11             if ¬ currentAction.isExecuted() then
12                 requestedSign ← worstRequest.sign;
13                 AVT.updateParameterValue(requestSign);
14                 currentAction.executed ← true ;
15             else
16                 currentAction.remainingSteps--;
17             end
18         end
19     end
20 end
```

With the addition of these mechanisms in the behavior of parameter-agents, the same test case as in the beginning of this subsection is ran. Execution log is depicted in figure 5.20

We observe the exact same behaviors in the first two phases. At that point, the delay is estimated, as described in the algorithm 5.5 of this section, and therefore, at step 14, even though requests are received, $p1$ chooses to wait 2 steps. This happens two times, before it goes beyond the optimal value and opposite requests are received. By proceeding this way, $p1$ gets closer to the optimal value, and **finally converges towards the optimal** before being completely stabilized around step 40.

Figure 5.20 — Test case 7: Test case 1 without assumptions on the execution order of the agents, and with the delay estimation mechanism



The simplest case of delay (communication delay) is handled properly. We now propose a more realistic test case, in which the delays result from parallel asynchronous activities between two systems.

5.4.2 Test Case 8: Asynchronous Systems

To analyze situations in which the delay between agents' actions and perceptions is more significant than what we have previously described, we here propose a test case with a slightly different structure.

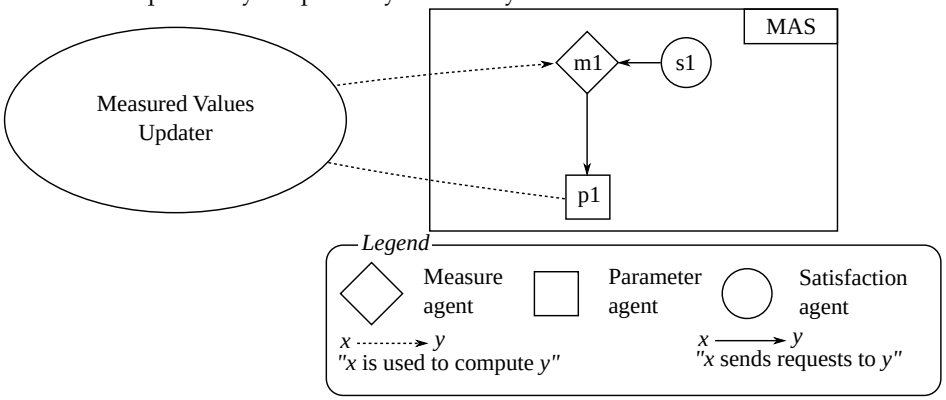
We consider a now familiar test case with a single parameter $p1$, a single measured value $m1$, and a single objective on this measure $s1$. We consider $p1$ to be able to take any value in the $[0, 100]$ interval, and we consider the same criticality function centered on 25 for $s1$ depicted in figure 5.4.

The difference is that we propose to run, next to the multi-agent system, a separate system responsible for the update of the measured values. We call that separate system the *measured values updater* (see figure 5.21).

The idea is that the multi-agent system runs on its own, as usual, with for instance a single step for all the agents every 500 milliseconds. That means that every 500 milliseconds, each agent is executed once, and observes its environment, potentially sends requests or changes its value. The *measured values updater* runs on its own. In our case, every 1500 milliseconds, it takes the parameter values it needs (in this case, the value of $p1$), calculates the values of the measured values (in this case $m1 = p1$), and sends them to the multi-agent system.

This implies that even if a parameter-agent has done the optimal action, it may need to wait an undetermined time to actually observe the consequences of its actions. And if in the meantime, the agent still modifies its value, then the system may produce oscillations as we

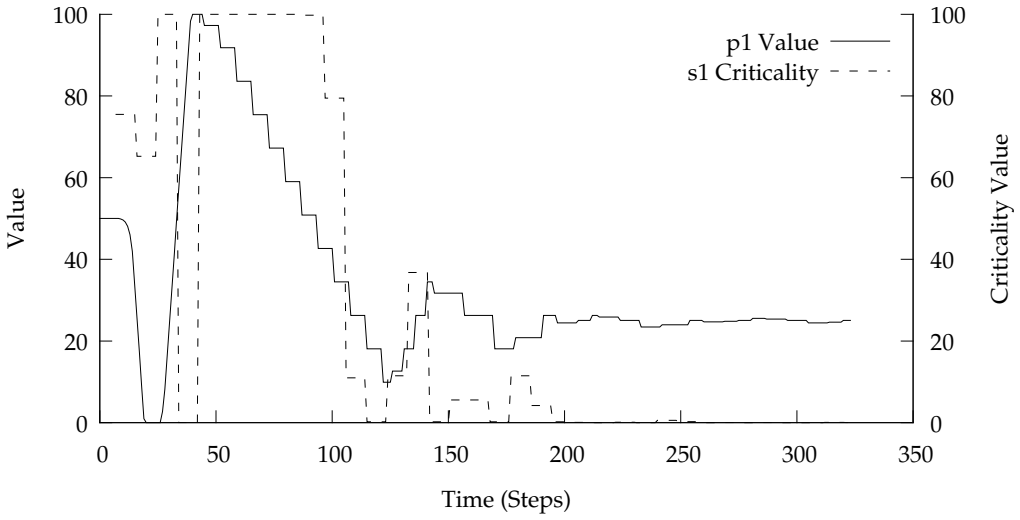
Figure 5.21 — Test case 8: one parameter, one measure and one objective, with the measured value updated by a separate system not synchronized with the MAS



have seen at the beginning of this section.

The test is ran with this *measured values updater*, and the execution log is depicted in figure 5.22, with the multi-agent system steps on the x-axis.

Figure 5.22 — Test case 8: Test case 3 but with an asynchronous process responsible for the update of the measured value



In the first 10 steps of the execution, nothing is received, therefore there is no activity in the MAS. Then, $m1$ receives its first value, and triggers activity. Consequently, requests demanding to lower the value are received by $p1$. Since no delay value is available at that point, the value is severely decreased, until it reaches 0 around step 25. Even though it went beyond the optimal value, because of the delay between the two systems, no requests demanding to increase the value is received.

After some time, around step 30, a request demanding to increase the value is finally received by $p1$. Since still no delay value is available, $p1$ increases its value several time without waiting. Similarly to the previous phase, even though $p1$ goes beyond the optimal value, no contradictory requests are received. When $p1$ reaches the maximal value, it stays there.

After some time, it finally receives a request demanding it to decrease the value, around step 45. Except now, a delay value is available. Consequently, the value is decreased once using the adaptive value tracker, and then $p1$ waits for the consequences of its actions to be effective. The same phenomenon is repeated several times until an opposite request is received around step 125.

Since $p1$ now waits for $m1$ to be updated, $p1$ does not increase its value indefinitely and reaches the maximum. Instead, it goes slightly beyond the optimal value, and waits for contradictory requests. By doing so, the distance between $p1$'s actual value and the optimal value it could take is gradually decreased, and it finally converges on the optimal value after 250 steps.

The time it takes for the parameter to converge may seem long, compared to the previous test cases, but we must note that **it is not the multi-agent system that takes time here**, but it is the system to be controlled (here the measured values updater) that is slow to react. Whatever the speed of the multi-agent system, the solving process is always limited by the target system. In our case, the multi-agent system is idle most of the time.

5.5 Conclusion

The various problem configurations we have detailed in this section are those we believe to cover all the situations in which agents may be. By focusing on local perceptions of agents, we believe **we are able to tackle the great complexity of problems we address**. We do not try to understand all the side effects of the actions agents take, or what consequences they should anticipate due to the complexity of their environment. The agents are focused on a simple task: maintaining cooperative interactions with their environment.

It is likely that agents will be confronted to problems that exceed the complexity of those presented here, but we claim that these problems can be in fact decomposed into these local problems, where interactions between constitutive parts of the system (i.e. agents) have antinomic activities. If agents manage to locally restore the cooperative nature of their interactions with their environment, the system as a whole can again be functionally adequate.

In order to verify that the designed multi-agent system is able to dynamically adapt parameters in order to reach objectives, we will apply it to various case studies, detailed in the next section. Through these case studies, we will investigate various properties that we believe to be necessary for a game adaptation system to be satisfactory.

6 Experiments and Validations

In the previous chapter, we have shown how the game adaptation system behaves in several test cases. Through these test cases, we have investigated representative configurations of various problems that may occur when the multi-agent system is confronted to various problems. Though we believe that all major situations are covered by these test cases, **we now need to apply the system to a greater variety of problems**.

By applying the multi-agent system to larger problems, we can demonstrate additional properties that we claim to be necessary for such a system to be satisfying. Among these properties are scalability, handling of complex dynamics, fault tolerance and applicability to complex domains.

In this chapter, we present several applications, from computer-generated problems, highly dynamics models, actual games and real-life application studies. All these applications set the focus on different properties, and combining all of them allows us to confirm the ability of the presented system to deal with the majority of problems in which it may be useful.

The first section presented here investigates the scalability of the multi-agent system, by proposing a way of generating a large quantity of problems of various sizes.

6.1 Computer-Generated Problems

An important property of a system dedicated to the real-time adaptation of parameters of systems like modern video games is the **scalability**. Indeed, if a system were only able to effectively adjust a pair of parameters accurately to satisfy given constraints and objectives, it would not be a a valuable help to handle real-life video games. Video-games usually feature a large set of parameters, each of them representing a precise aspect of the game. When many artifacts and rules are involved in a game, each of them may have several characteristics, and each of these characteristics is likely to have an influence on the game experience.

Therefore a game adaptation system should be able to handle such amounts of parameters, and the process of finding adequate values should **still take a reasonable time even when dealing with larger or more complicated problems**.

In this section, we propose to conduct a statistical analysis on the scalability of the pro-

posed system. For that purpose, we propose a way to **procedurally generate a large quantity of problems**, with various degrees of difficulty or complexity. By using an automatic generation of problems, we can launch the adaptation system many times, and we can conduct an analysis of the impact of the size of the problem on the resolution process.

In this section, we start by describing the way we generate problems, then we detail the methodology we have used to run resolution processes, and finally we comment the obtained results.

6.1.1 Problem Procedural Generation

Problem generation is an efficient way to test the proposed system on a larger set of problems. By generating problems with a complexity (in terms of parameter-space size, inter dependencies and delay effects) equivalent to real life problem, **we can verify that the proposed system is able to scale up**. The fact that these problems are not exactly of the same nature as the problems we want to address in this thesis does not make the testing irrelevant, since the system we design is adaptive, and must therefore be able to handle problems without being explicitly designed to deal with them. Besides, this approach has already been successfully taken to design other control systems [Boes, 2014].

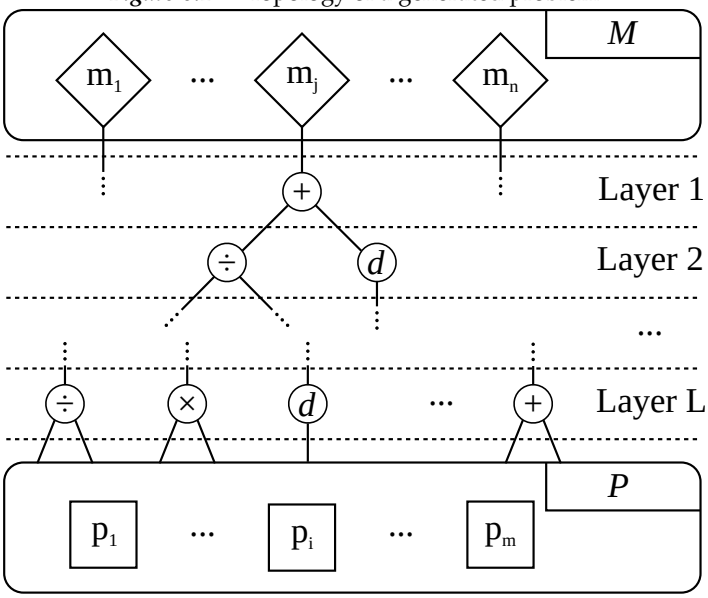
In order to generate problems, we start by defining a set M of measured values m_j, on which we will later define objectives to reach. At the beginning of the generation process, we only define the number n of measured values that we want to include in the generated problem. The procedural generation of the problem then consists in building the conceptual structure that will generate values for these measures.

To build this conceptual structure, we use a set of operators that can be combined to form more and more complex calculations trees. We propose to use three of the main arithmetic operators: sum (denoted +), product (denoted *) and division (denoted ÷). These operators have a standard behavior: they take two input values (operands), and yield a single result value each time they are executed.

To create more realistic problems, we introduce another non-standard operator: the *delay* operator (denoted d). The goal of this delay operator is **to introduce variability in the time before consequences of parameter changes are actually observed**. For this operator, we propose to use the following behavior: a delay operator has a single operand, and each time something requests a value from this operator, it yields the mean value of the last 30 inputs it received. We consider a received input to be the value of the input at the moment the operator is asked for a value. As a consequence, if the input of the operator changes to a value x, it will take up to 30 requests before what is yielded by this operator is actually x. 30 is an arbitrary number, unknown by the game adaptation system. It is large enough to cause perturbations in the game adaptation system, as seen in section 5.4.

Operators can be combined with each other as many times as wanted to produce new values. We propose to organize operators in consecutive layers (see figure 6.1). Each measurable value m_j in the previously mentioned measurable values set M creates an operator in the first layer to compute its value. The operator is randomly chosen between the four possible operators.

Figure 6.1 — Topology of a generated problem



Each of these newly created operators needs one or two operands to be able to actually compute a value. Therefore, each time a value is needed by the operators of layer 1, another random operator is created in layer 2 to yield that value. An arbitrary amount of layers can be generated following this process, multiplying the number of needed values by up to two with each new layer. Let L represent the number of layers. When enough layers have been generated, there are up to $2^{L-1}+1$ operators that need operands for each measurable value; these operands are parameters.

At the beginning of the generation process, the set P of parameters is empty. Each parameter is identified by its index, and characterized by its current branching factor, which has a predefined upper limit. Parameters have a maximal branching factor they cannot exceed. Therefore, when every parameter in P has reached its maximal branching factor, operators are not able to find an available parameter in P, and thus, a new parameter needs to be created. These parameters are arbitrarily defined on the $]0,5[$ interval.

An additional rule in the generation process states that all parameters can be used only once in the calculation of a given measured value (but one parameter can be used in the calculation of several measured value).

To summarize, the topology of a generated problem is characterized by three key values:

- the number of measurable values in the set P,
- the number of layers of generated operators,
- and the maximal branching factor of the parameters.

Aside from these three values, a random generator is used to determine the generated operators, and to select the parameters in the last layer.

The same random generator is then used to give all the parameters $pi \in P$ random values

v_{p_i}. These random values are used to calculate the current values v_{m_j} of all $m_j \in M$. These values v_{m_j} are selected to be considered as the optimal values for all the measurable values.

Objective values for all measurable values are now known, and we are sure that these objectives are all attainable at the same time. The values of the parameters can now be shuffled: random values v'_{p_i} are assigned to every parameter $p \in P$, and as a consequence, all the new values v'_{m_j} are now different from their objectives.

The goal of the game adaptation system is now to modify the parameters in order for the measurable values to reach their objectives.

6.1.2 Application of the Game Adaptation System

When dealing with generated problems as we have described them in the previous section, the goal of the game adaptation system, as described in the previous chapter, is to modify the parameters $p_i \in P$ in order for the measured values $m_j \in M$ to satisfy their objectives. Thanks to the way the objectives were determined, we are guaranteed that a solution exists for any generated problem.

For each parameter $p_i \in P$, we create a parameter-agent. Each parameter-agent is named after the index of the parameter, and has the ability to modify its value.

For each measured value $m_j \in M$, we create a measure-agent, named after the index of the measured value. These agents are responsible for maintaining their measured value up to date. In other words, they are aware of the generated operators, and, at each step, they use them to compute the values of their measure.

When constructing the problem graph, we list all the dependencies from all the measures to parameters. Therefore, when creating the measure-agents, we make them aware of the parameter-agents that are likely to trigger a variation of the measured value they are responsible of, and they know the sign of this correlation. As described in the previous chapter, measure-agents have the ability to send requests to these parameter-agents.

For each measured value $m_j \in M$, we create a satisfaction-agent s_j that represents the objectives defined on measured values. By construction, we know the bounds $[min_{j_min}, max_{j_max}]$ of each measured value m_j. Thanks to this information, we create the criticality functions for each satisfaction-agent s_j so that:

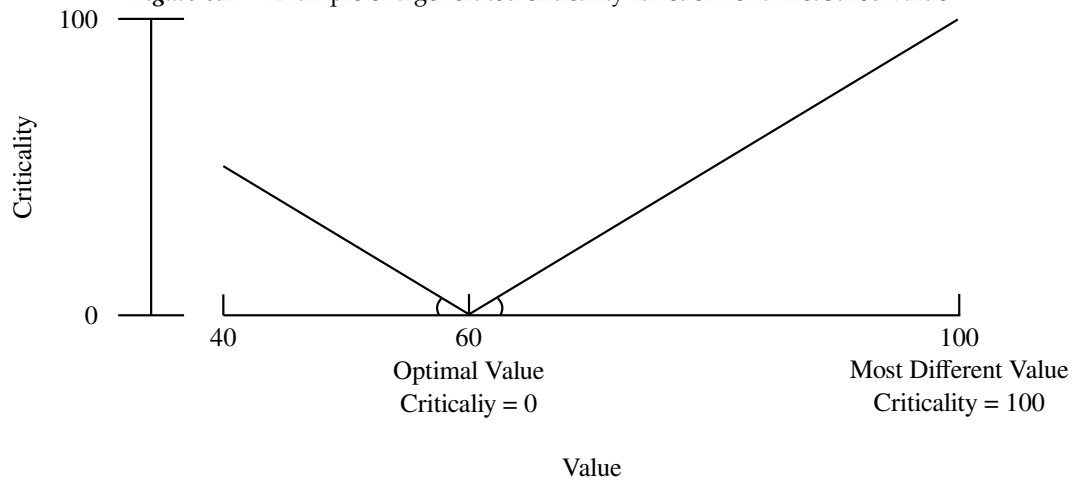
- The optimal value has a criticality of 0.
- The more different the current value is, the higher the criticality.
- The value the most different from the optimal value is assigned to a criticality of 100.

An example is shown in figure 6.2. The interval of the value is $[40, 100]$, the optimal value is 60. Therefore, the criticality function is null at 60. The value the most different from the optimal is 100, and consequently, the criticality at this point is 100, and all the other criticality values are assigned accordingly.

As described in the previous chapter, satisfaction-agents send requests to measure-agents, which, in their turn, send requests to parameter-agents, which may or may not

modify the value of the actual parameters, as explained in the description of their behavior.

Figure 6.2 — Example of a generated criticality function for a measured value



6.1.2.1 Scalability Experiments

In a classic way, in order to test the scalability of the proposed game adaptation system, several experiments, based on the same problem, must be run. Before each run, the difficulty of the problem must be increased. For each run, how long it took to find a solution must be registered since this criterion enables determining the scalability of the proposed approach if the resolution time does not grow exponentially when the difficulty is increased. Several difficulty levels should be proposed in order to have enough data to conclude about the ability of the game adaptation system to solve large problems within a reasonable amount of time.

Consequently, we first need to determine what makes a problem, generated as described in the previous section, more difficult than another one.

The first idea is that the difficulty increases with the number of measurable values, each of them having its own objective, as well as the number of layers (and thus, the number of parameters). However, this idea is not completely true. For instance, when generating a problem with 20 measurable values and 7 layers of operators, the number of parameters needed by a single measurable value is going to be up to $2^L = 2^7 = 128$, and potentially less than that, given that the delay operator is unary.

While this may be an important number of parameters, it does not imply that the problem is difficult. The important notion here is the maximal branching factor of the parameters. If the maximal branching factor is 1, then each measurable value in M is going to generate its own set of parameters in P. As a result, P is going to be composed of up to $20 \times 128 = 2560$ parameters. Again, that number may seem important, but in that case, each parameter only affects a single measure-value. Therefore, when the multi-agent system is run, each parameter-agent will receive requests coming from a single measure-agent, which will result in fairly simple configurations. It would be like having 20 completely in-

dependent problems being resolved next to each other without any interference. Still, this experiment could be run to determine if **the system is able to handle several thousands of parameters at the same time**.

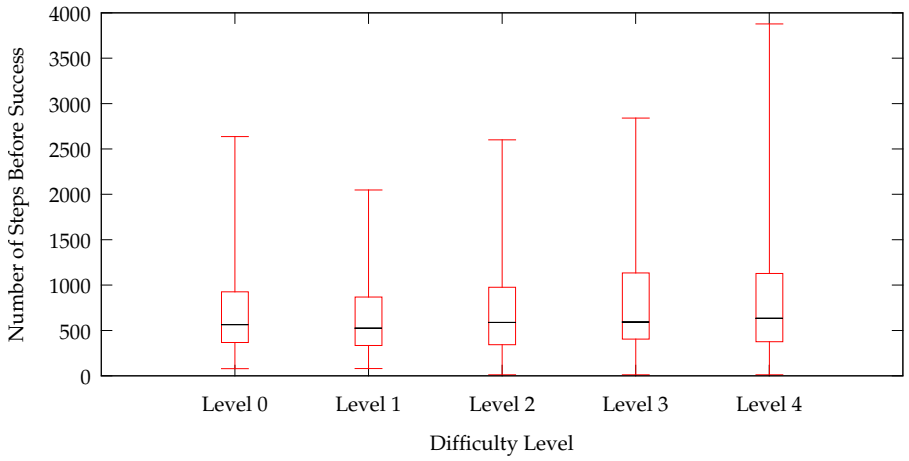
A more interesting approach would be to progressively change **the maximal branching factor**. If the maximal branching factor is 2, there is half as many parameters than in the previous experiments, but each measured value is now competing with another one regarding the modification of the parameters. By keeping increasing the maximal branching factor, we increase the difficulty: **the higher the maximal branching factor, the more the problems of the measured values are interdependent**, and thus, the more difficult a problem is.

We propose to run experiments with **an increasing degree of difficulty**. For that purpose, we define **5 levels of difficulty**. Every level consists of 20 measurable values, that are calculated with 5 layers of operators. What changes between these levels is the maximal branching factor. At level 0, the maximal branching factor is 1. Consequently, the problem is quite easy, as the 20 measurable values are completely independent. At level 1, the maximal branching factor is 2. At level 2, it is 5, at level 3 it is 10, and at level 4, it is 20, which is the maximal value. This means that **at level 4, all the parameters are involved in the calculation of all the observable values**.

For a single experiment, we launch the resolution process on 100 generated problems, as described in figure 6.1, with the same generation parameters. This way, the obtained results will be more statistically significant. Figure 6.3 shows the results obtained in the 5 levels. We observe that even though the difficulty of the problem is increased more and more rapidly as the levels go up, **the resolution time is not significantly increased**, which is encouraging regarding the scalability of the proposed system.

The dynamics of the resolution time can be observed by looking at the evolution of the boxes, rather that the evolution of the whiskers, that represent single problems, and thus, do not have statistical significance.

Figure 6.3 — Results of 5 sets of 100 experiments with augmenting degrees of difficulty

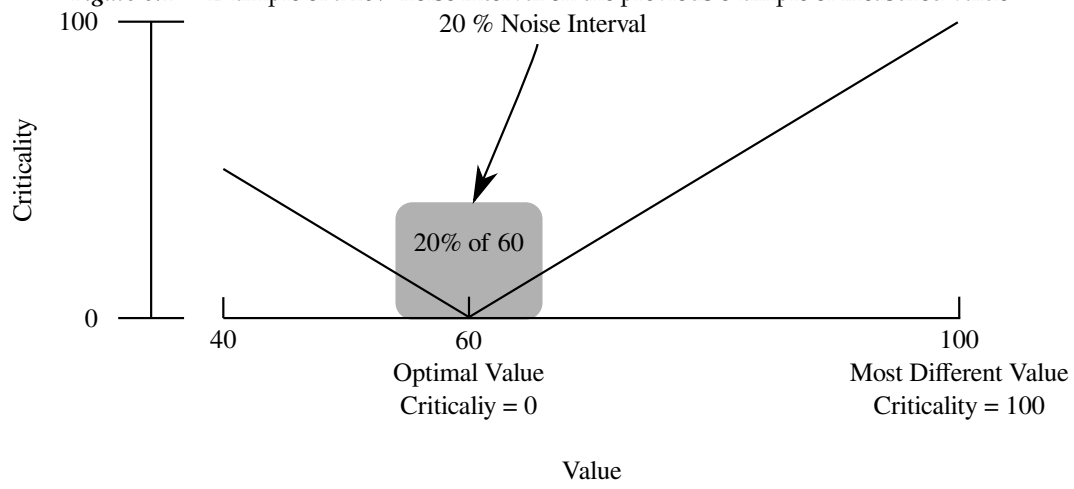


6.1.2.2 Noise Resilience, or Graceful Degradation Experiments

The second aspect that could be interesting to investigate is the ability of the game adaptation system to solve problems in a noisy environment. Noise occurs when unwanted information is received by a system, altering the legitimate information it needs for its nominal activity. A poorly designed system is likely to be completely unable to handle noised information, and produce unwanted results. The property of fault tolerance of a system implies that if the information is absolutely necessary to solve the problem, then the quality of the system should only be altered and the system should not completely fail. This is also referred to as a state of *graceful degradation*.

In order to test the ability of the proposed system to handle noise, we suggest to **deliberately alter the information perceived by the agents** in the system. As an analogy to actual systems, we propose that the measured values we observe on the system are altered by noise. Consequently, when measure-agents get their values from the generated problem, we add a given quantity of noise. Noise, in this case, is random information added to the actual information. To conduct our analysis, we modify the quantity of noise perceived by measure-agents. For that purpose, we introduce the noise rate, expressed in percentage, that determines how far away from the actual value the measure will be perceived by the measure-agent, as depicted in figure 6.4.

Figure 6.4 — Example of a 20% noise interval on the previous example of measured value

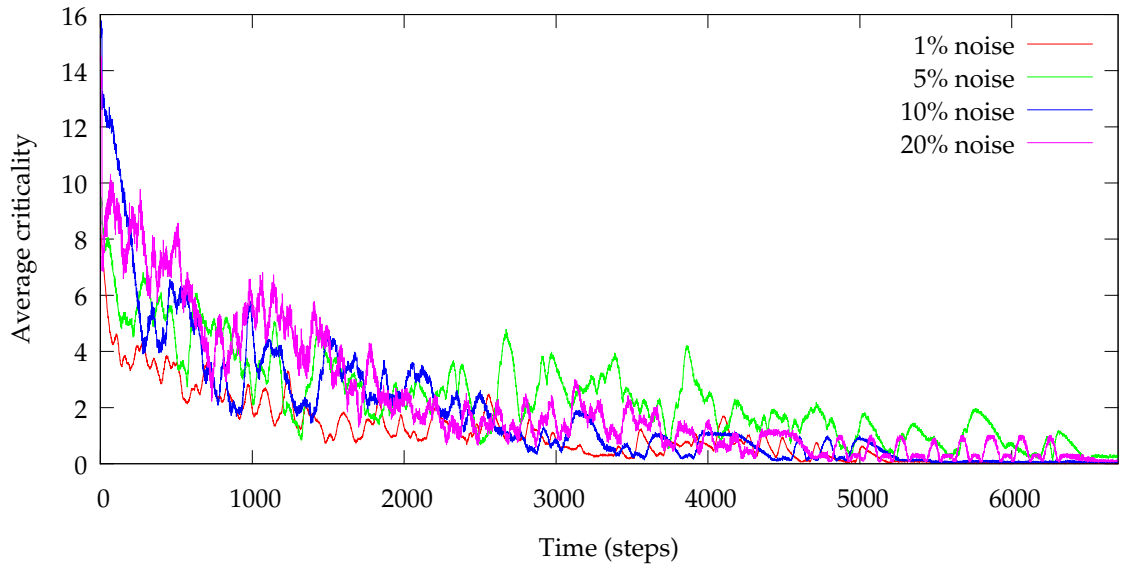


Of course, the system is not expected to find the optimal solution when the information it perceives is noised. However, a robust system should be able to provide partially satisfying results, and not be completely unable to function due to a small proportion of noise. In other words, the system should be able to find values for parameters around the optimal, with a certain degree of accuracy. The more important the noise, the less accurate the system is expected to be.

To determine if the system is robust to noise, we can therefore increase the noise rate, and run the system. We claim that the system should be able to find the optimal solution, as precise as the noise interval. Therefore, the system should be able to remain in a state where the global criticality is not greater than the highest criticality around the optimal value.

We have run experiments to determine how the system behaved with the addition of noise. We have selected the highest degree of difficulty, that is, where the maximal branching factor is of 20, and therefore, all the parameters are involved in the calculation of all observables. We have selected different noise levels: 1%, 5%, 10% and 20%. For each of these noise levels, we launch the system on 100 randomly generated problems. Figure 6.5 shows the evolution of the maximal criticality levels on these experiments. For each noise level, at each step in time, we calculate the average criticality level on all the experiments. We can therefore observe how the problems are generally solved.

Figure 6.5 — Execution log of 100 experiments at different noise levels



We observe on this figure that even though the measures are noised, **the dynamics of the system still makes the global criticality decrease over time**. After 6000 steps, all the experiments have converged to a criticality level close to 0. Though, the noise prevents the criticalities from being completely null, the system manages to **keep them stable at low levels**. In other words, **the noise does not prevent the system to converge towards to optimal values**.

6.1.3 Conclusion

With these experiments, several properties were investigated. First, the fact that these experiments were procedurally generated allowed us to run many similar experiences automatically in order to obtain statistical significance.

The first set of experiments demonstrated that the approach is able to tackle problems with a large number of parameters and observables. In order to solve such problems, the system just needs to create agents for each new element of the problem. Even with a large number of elements, **the activity of a single agent does not grow heavier**, as all it does is observing its environment and sending requests, or receiving requests and modifying its own value. Whatever the number of elements, the behavior of the agents is not changed, nor is the complexity of their reasoning. Only the number of agent is increased. Therefore, the

computing power needed to run the system **is only proportional to the size of the problem, and does not explode when the problem gets more complicated**.

We also investigated what happens when the difficulty of the problem is increased, which may not be the case when the number of elements increases, since all these parameters and measures may just consist of independent problems. In order to increase the difficulty of the problem, we made sure that all parameters were involved in the satisfaction of all measured values. **As the difficulty of the problem increased more and more rapidly, the resolution time only progressed slowly**.

The second set of experiments demonstrated that even though the information on which the system is based to perform adaptation is noised, **it is able to perform in an adequate manner**. Of course, perfect solutions are not attained if the system does not have complete information, but the best is obtained from the available information, and the global criticality is almost null. In other words, when information is altered, the performance is only altered proportionally, and not totally.

These experiments reinforce the idea that the proposed system is able to solve problems with a great complexity, even in environments where only incomplete or noised information is available.

6.2 Dynamical Equilibrium – The Prey-Predator Model

Another aspect that we consider to be necessary for a parameter adaptation system in our context, is the ability to **handle highly dynamic environments**. Dynamic environments can represent several things. First, the environment in which the system evolves can be dynamic, in the sense that it **changes spontaneously**, without the intervention of external entities. The consequence is that even if parameters values that satisfy all objectives are found, they may not remain optimal over time, as the system changes by itself, forcing the adaptation system to constantly update the parameter values in order to keep the objective satisfied.

The second aspect of a dynamic environment involves the requirements given to the adaptation system. Requirements, or objectives values that measures of the system activity should satisfy, are typically expressed by humans. They represent the desired orientation of the system. Like everything expressed by humans, they are prone to change over time. **Objectives may not remain the same in the whole life-cycle of the system**.

In order to experiment on these aspects, we propose to apply the approach on a model that is, by nature, highly dynamic: a prey-predator populations model. Though many variations exist, the principle always remains the same. An ecosystem features two species that interact with each other: preys, that reproduce spontaneously, and predators, that consume preys in order to reproduce.

6.2.1 The Prey-Predator Model

Prey-predator models have been extensively studied, and they are generally expressed in the form of two differential equations, attributed to both Lotka and Volterra, and thus, referred to as Lotka-Volterra equations [Lotka, 1924; Volterra, 1927]. These two equations calculate the population of the species of preys and predators based on several parameters. They have been used to describe the evolution of numerous biological systems, and have been studied in various fields, such as biology, mathematics and economics. These equations can be expressed as follows:

$$\begin{cases} \frac{dx(t)}{dt} = x(t)(\alpha - \beta y(t)) \\ \frac{dy(t)}{dt} = -y(t)(\gamma - \delta x(t)) \end{cases}$$

Where

- t is the time,
- $x(t)$ is the amount of preys at time t,
- $y(t)$ is the amount of predators at time t,
- α is the reproduction rate of the preys,
- β is the death rate of preys due to encounters with predators,
- γ is the death rate of the predators (independent of the amount of preys), and
- δ is the reproduction rate of the predators according to the preys encountered and consumed.

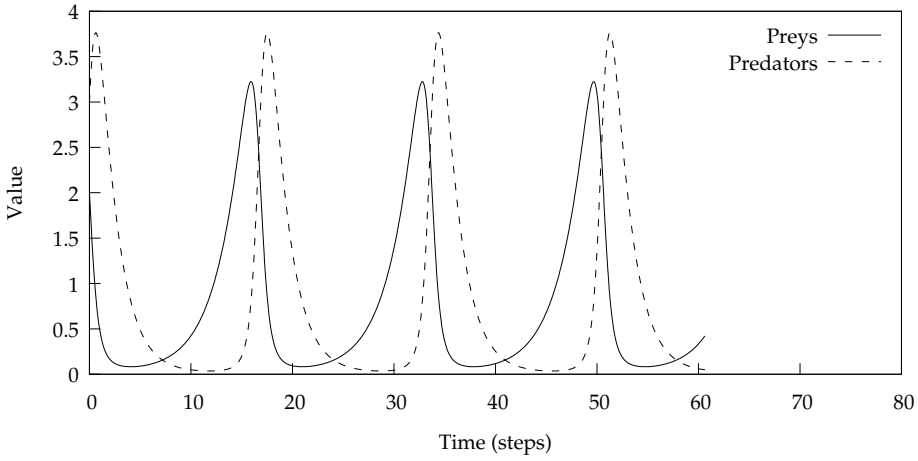
The two equations represent the variation of the two populations over time, parametrized by four values: α, β, γ and δ. These variations are not necessarily constant over time, as they depend on the levels of the species populations. The growth of preys is exponential, the more preys there are, the more they will reproduce and, at the same time, the more predators will be able to proliferate.

This prey-predator model is a good example of a dynamic nonlinear system. Depending on the values of the four parameters and the initial conditions, several patterns can be observed. First, there are many cases where the system is not able to stabilize, where, for instance, the predators consume all the preys, and subsequently go extinct due to the lack of food. In other cases, predators may not be able to reproduce fast enough to maintain a positive population, and go extinct, letting the preys reproduce indefinitely.

Other cases, more interestingly, allow the two populations to coexist, but remain highly dynamic, with the two populations oscillating. Figure 6.6 shows an example of dynamic equilibrium. Population levels are never stable, but they are able to alternate peaks and valleys to both remain non-null over time.

Another way to look at the system is to omit the time from the graphical representation. A chart may represent the amount of preys on an axis, and the amount of predators on the

Figure 6.6 — Dynamic populations equilibrium in the Lotka-Volterra model



other. The previous example is depicted this way in figure 6.7. The elliptic-like shape shows that the behavior of the system is cyclic, and the amounts of populations limited to a finite set of values.

What can be observed on this system is, that if the initial conditions are changed, another dynamic equilibrium occurs. Figure 6.8 shows the same equations with different initial conditions. These curves are centered around a single point, which corresponds to a state where the variations of both populations are null. This point has been found analytically, and can be expressed as: $(\frac{\gamma}{\delta}, \frac{\alpha}{\beta})$. For a given system (that is, for given parameters values), there is a single point like that, where the variations are null and the populations are constant over time.

However, this equilibrium is unstable, meaning that if a population is changed, a constant state cannot be reached anymore, and the system will oscillate in an elliptic-like shape around the center.

Figure 6.7 — Predator population according to prey population

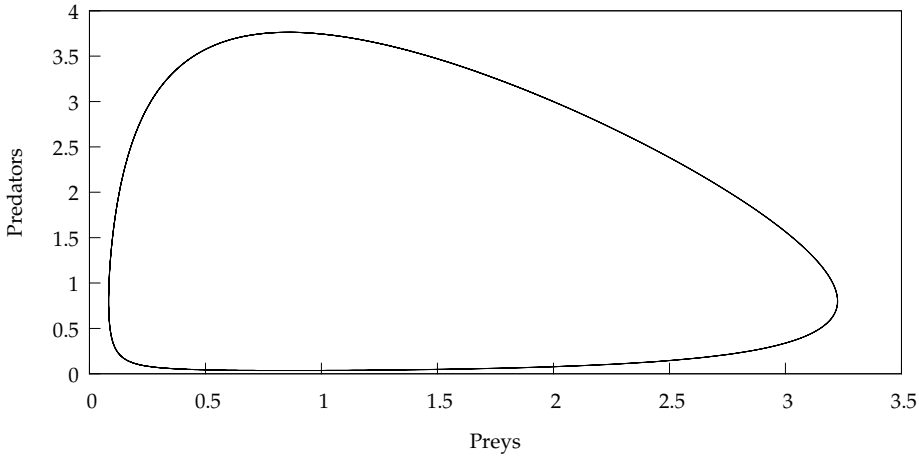
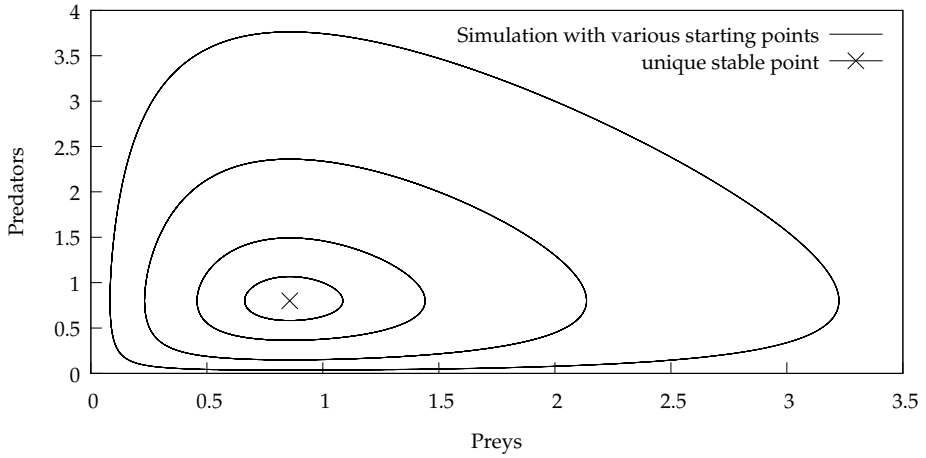


Figure 6.8 — Predator population according to prey population, with various initial conditions



6.2.2 Applying the Game Adaptation System to the Prey-Predator Model

The prey-predator model is a great example of dynamic system. Populations levels are measurable values and, except for a very specific state, they change spontaneously. Parameters like the death and reproduction rates influence the evolution of the populations, and direct correlations can be intuitively found.

An interesting application of the game adaptation system to this model can be to modify the reproduction and death rates in real-time, in order for the two population levels to reach predefined objectives. The populations objective should not necessarily be the levels at which the system is stable and the populations are constant.

Moreover, since we want to investigate the ability of the system to handle dynamic environments, it would be interesting to **dynamically modify the objectives** expressed on the population levels.

To run all these experiments, we first need to design a simulation able to manipulate population levels, and to make them evolve over time. A simple program that manipulates the equations is enough. The program declares two variables x and y to represent the two populations, and then calculates an approximation of the evolution of the populations on a single time step. To gain accuracy in the numerical estimation, the time steps should be chosen small enough to not induce too much variations from the formal equations.

However, even if the numerical estimation of the evolution of the populations is not accurate, it does not impact the interest of our experiments. We could always consider that if we only have a numerical model based on the Lotka-Volterra equations, it would still have the dynamical properties we are interested in. No matter how they are calculated, the two variations of populations are added to the current populations to update them, and the operation is repeated indefinitely.

Along with this prey-predator simulation, we can instantiate the game adaptation system that will interact with it. Parameter-agents are created for the four rates involved in the system. In other words, a parameter-agent is created for α, β, γ and δ. These parameter-

agents can, as always, modify the value of their related parameter, which is, according to the model, always in the interval $[0, 1]$.

Two measure-agents are created for the population levels. Each of these measure-agents is responsible for monitoring the actual population level computed by the simulation. To represent the variations of the populations, we also create two measure-agents, one for each measured variation.

Finally, two satisfaction-agents are created to encapsulate objectives on population levels. The objectives are numerical values, that determine population levels we want to observe in the simulation. A satisfaction-agent is created for each objective which observes the value of the corresponding measure-agent.

The correlations between agents are straightforward. The measure-agents representing the population levels are correlated to their respective measure-agent representing the population variation. Then, the measure-agents representing the variations are correlated to the parameter-agents as depicted in table 6.1.

Table 6.1 — Correlation matrix for the prey-predator model

	Prey Var.	Predators Var.	Prey Pop.	Predators Pop.
Prey Var.			+	
Predators Var.				+
alpha	+			
beta	-			
gamma		+		
delta		-		

Once these agents are instantiated, it is possible to conduct experiments, which are now detailed.

6.2.3 Experiments

The first experiment consists in running the simulation while expressing two requirements on the two population levels. In our case, we arbitrarily choose to set the requirement on the predator population to 15, and the one on the population of preys to 5. These requirements are expressed with the help of the criticality functions (see section 4.2.3) shown in figure 6.9.

Arbitrary initial values are chosen for the two population levels (preferably different from the objectives), in this case, 2 is chosen for the two populations. Then the simulation is run along with the game adaptation system.

Figure 6.10 shows the execution log of this specific simulation. We observe that the two population levels progressively increase until they reach their objective value around step 2000. Small dynamics are still observed, but the values remain close to their objectives. When observing the evolution of the criticalities of the two population objectives (figure 6.11), we see that these criticalities decrease steadily, even though slight variations are observed, to finally remain null after step 2000.

The two graphs in figure 6.12 demonstrate that the behavior of the system is no longer

Figure 6.9 — Criticality functions expressing the two objectives on preys and predators populations

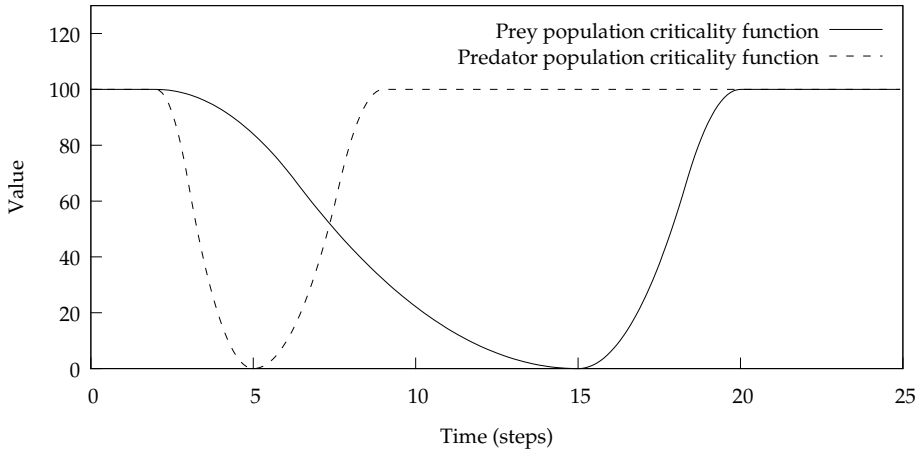
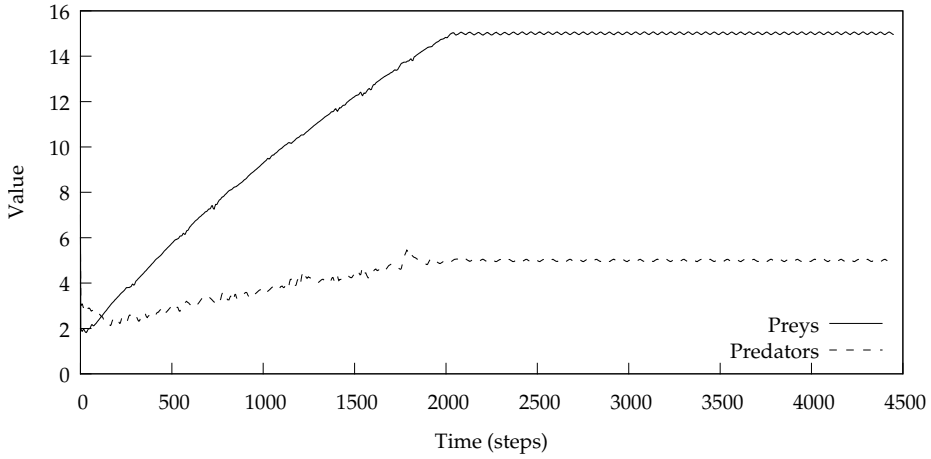


Figure 6.10 — Execution log of the first prey-predator simulation with objectives 5 and 15



cyclic, but stable. The left part (a) depicts the number of predators in function of the number of preys in the whole experiment, whereas the right part of the figure (b) shows the same values but only after step 2000. The first figure shows us that from the initial conditions, the system is moving toward the objective, and the second figure shows that the system is stable once it has reached the objective.

With the second experiment involving the prey-predator simulation, we try to investigate the second aspect of what makes a system dynamic, by changing the objectives of the populations as the system is running. We propose to start the experiment similarly to the first experiment. Figure 6.13 shows the execution log of the experiment.

Once the system is stabilized, we change the objective of the two populations. Around step 2700: from the value pair (5, 15), we change it to (8, 11). We change it again to (5, 16) around step 4000, and finally to (7, 13) at step 5500. We see that the population levels closely follow the objective, and are able to start evolving as soon as the objective changes.

Criticalities in figure 6.14 illustrate the same phenomenon. As soon as the objectives

Figure 6.11 — Criticalities log of the first prey-predator simulation with objectives 5 and 15

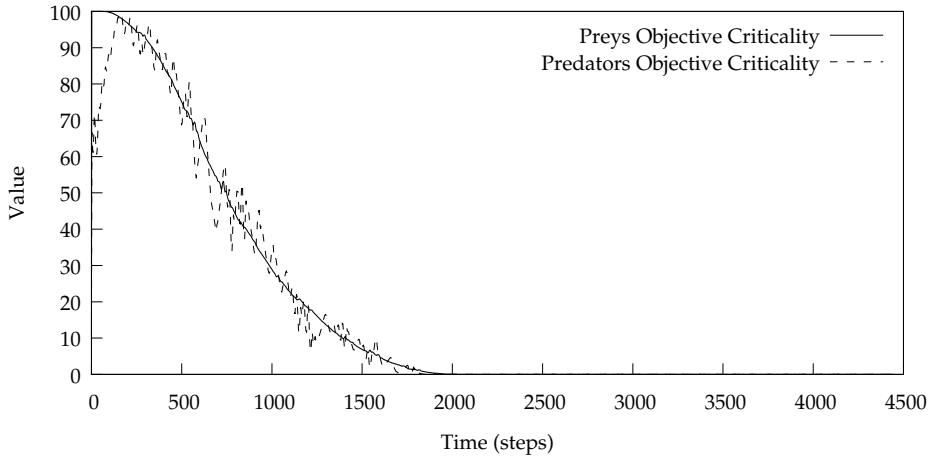
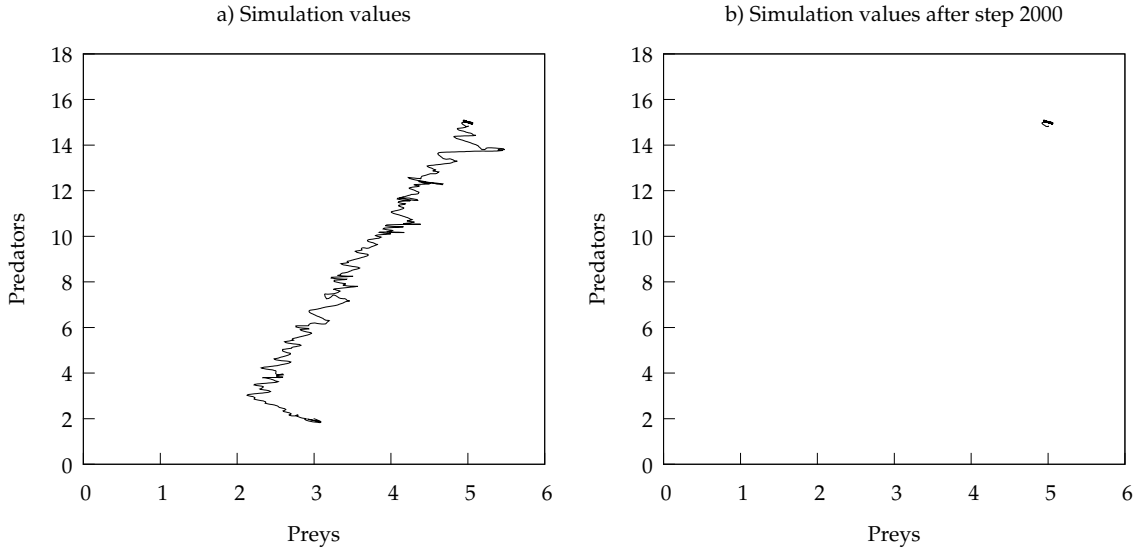


Figure 6.12 — Predator population in function of prey population, during all simulation (a) and after 2000 steps (b)



change, the criticality levels are suddenly increased. And as the population levels get closer to their respective objective, the criticalities are lowered, to finally be stabilized on approximately null values.

6.2.4 Conclusion

Given the results obtained in these experiments, we can state that the game adaptation system proposed here is **able to modify parameters in real time**, even if the target system is dynamic. The prey-predator model is a great example of system with nonlinear dynamics. Modifications of the parameters have different consequences depending on the context in which they occur. Moreover, parameter modifications in order to reach a certain goal may

Figure 6.13 — Execution log of the second experiment, with objectives varying over time

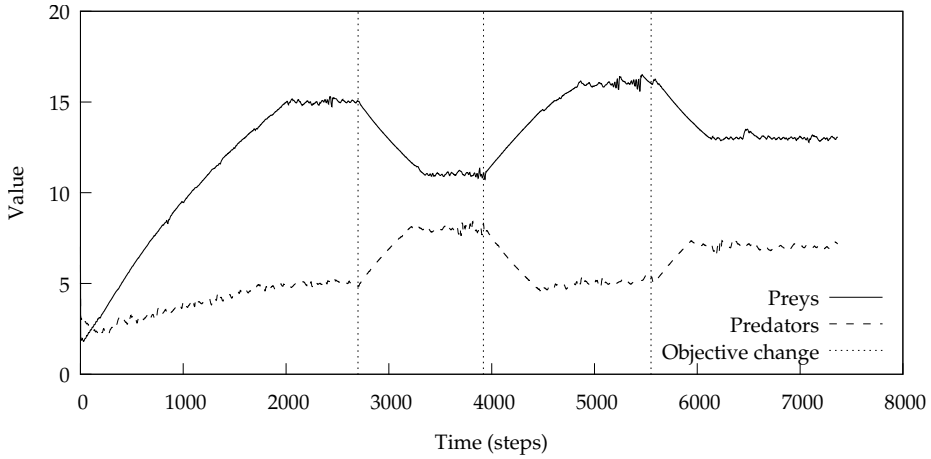
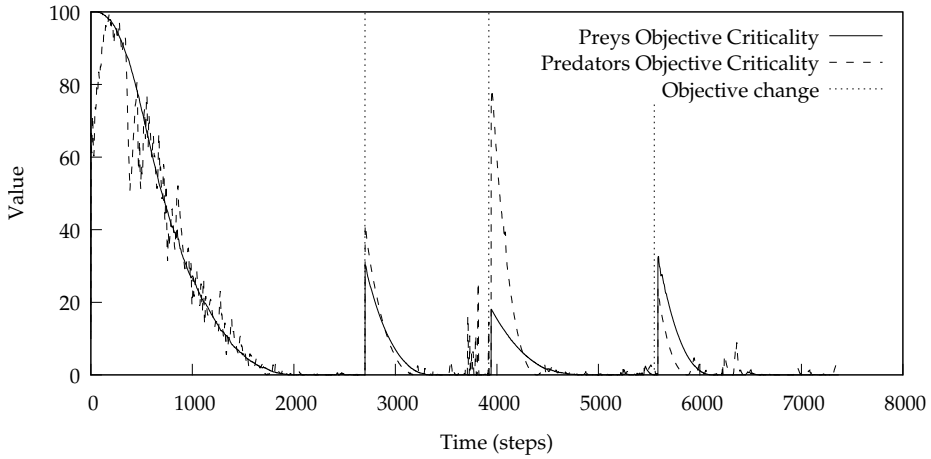


Figure 6.14 — Evolution of the objectives criticalities during the second experiment



have consequences that are counter productive for reaching that goal. But by considering only direct correlations **we did not need to anticipate the side effects** of certain parameter modifications, and **we were able to properly modify the parameters and reach the predefined objectives**.

The second part of the experiment explored another dynamic aspect of systems to be adapted. We observed that even though the parameter modification process was stabilized on optimal values, when the objectives were changed, the adaptation system was **able to modify the parameter again to find new optimal values**. Interestingly, finding the new optimal values did not take more time than finding the previous ones. This is an advantage of systems that do not actually learn, but instead, adapt themselves to their environment. When the environment changes, the system changes. **The fact that the system did not learn makes it able to quickly react, with no need to "unlearn" previous configurations**.

Overall, these experiments validated the idea that our game adaptation system is able to properly function even in highly dynamic environments. In the next sections, we apply the approach on actual games, to see how the approach fits in different contexts.

6.3 ASD – Tower Defense

In order to determine how convenient and how efficient the game adaptation system we have proposed in the previous chapter is, we propose here to use it on an actual game, so it is confronted to players in playing conditions. For that purpose, we choose to use the game *ASD – Tower Defense*[1] (ASD–TD).

6.3.1 Tower Defense Games

ASD-TD is a *Tower Defense* game, which is a popular genre, with plenty of examples online. This type of game usually takes place on a fixed map. Computer-controlled entities are spawned at a given spot on the map, and their goal is to move in order to cross the map. The goal of the player is to prevent these entities from crossing the map, by placing *defenses* on their path. Once a defense is placed, it acts autonomously, mainly by attacking the entities that are within its range.

What distinguishes a skilled player from a beginner, is the selection of placed defenses, and of courses, their locations on the map. A player has to consider several aspects before making a choice. There are usually several defense types, each with their own advantages and drawbacks. Similarly, there are several types of artificial entities, each with their strengths and weaknesses. Some defenses are efficient against certain types of enemy, and not against others. Some defenses need more resources to be placed, but are more efficient. Other consume less resources, but are only efficient against a specific type of enemy, etc.

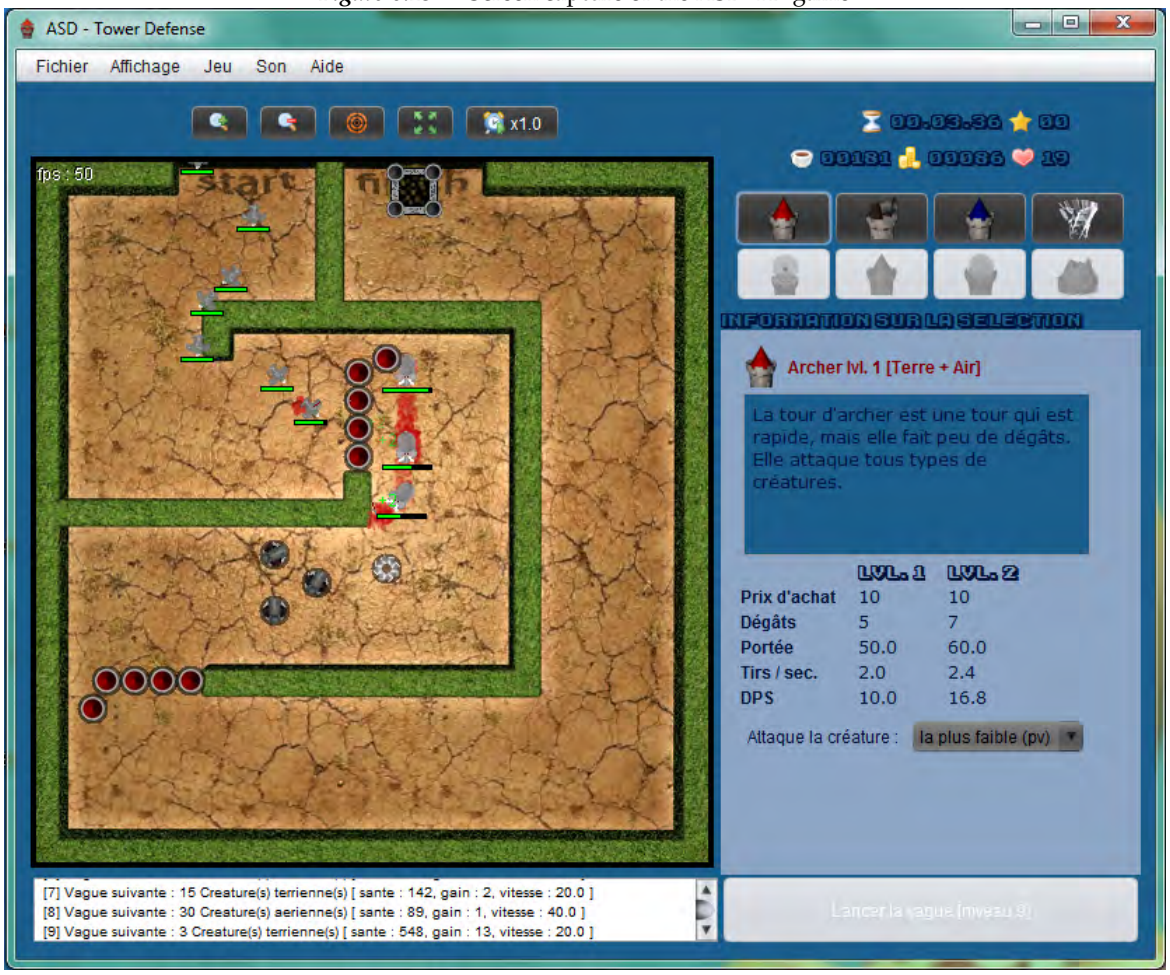
Artificial entities – or enemies – generally spawn in *waves*. A wave is a short period of time in which several entities are spawned and start to cross the map. The more enemies are grouped together, the hardest it is to efficiently prevent them from crossing the map. If enemies successfully cross the map, a penalty is given to the player. Depending on the game, there can be a penalty threshold, after which the player is considered defeated, and has to start over. Usually, when all the enemies of a given wave are defeated, there is a break period, during which the player can rest, and adjust its strategy (i.e. place different defenses at different spots on the map) before the next wave starts.

Most of these gameplay components are found in ASD-TD, and no specific game concept is proposed by the game. The main particularity of games is their artwork, and the story elements they propose to players to build a pleasant and fun atmosphere. A screen capture of the game is shown in figure 6.15. The map is shown on the left part of the screen, with enemies trying to cross it. Each enemy features a green bar indicating its remaining life. The right part of the screen shows the commands available to players. There are buttons to select defenses to place them on the screen, as well as indications on their characteristics.

When playing a game of this type, players can be placed in a large number of different situations. Many aspects of the game determine the game experience that a given player is facing. For each entity type, there are several parameters, such as the moving speed or robustness. For each defense type, there are also several parameters, such as the rate of attack, the effectiveness of the attack, or its range. The waves themselves can also be

[1] http://asd-td.com

Figure 6.15 — Screen capture of the ASD-TD game



parametrized, by defining the number and the types of entities present in the wave, as well as the frequency of entity spawning.

Before starting to apply the game adaptation system as described in the previous chapter, we must first list the parameters involved in the game in a more rigorous form. For this experiment, we consider parameters arranged in three categories. First, characteristics of the artificial entities that try to cross the map. We select three types of enemies, with different strengths and weaknesses: cheep, eagle and rhinos. Each enemy type has a speed and a strength. Speed is expressed in points per second, and strength in health points each individual of the given type has.

The second category is about the defenses that the player needs to manipulate in order to prevent enemies from crossing the map. In this experiment we consider only three types of defenses: regular, heavy, and light. These three defenses each have three parameters: strength, frequency, and range. Strength is the amount of health points a defense removes from an enemy when it attacks it. Frequency is the time it has to wait between two attacks, expressed in milliseconds, and range is the distance from which it can attack an enemy (expressed in points).

The last category groups together parameters of the waves that temporally characterize

how the game unfolds. Each wave has several characteristics: the number of individuals for each type of enemy, and the frequency at which enemies are spawned on the map. It is expressed as the pause time between two entities spawning.

All these parameters contribute to determine the difficulty level experienced by the player. Similarly to the majority of games, players can be facing challenges where the difficulty level is too high for them, resulting in frustration, or where the difficulty level is, by contrast, too low, and thus, resulting in boredom (this problem is described in chapter 1). The game adaptation system described in the previous chapter could be used to dynamically adapt the game experience so it matches the skill levels and abilities of players.

6.3.2 Applying the Game Adaptation System to ASD-TD

In order to dynamically modify the game experience to adapt it to players, the first step is to identify the inputs and outputs of the system. In other words, what can be modified, and what we want to observe. In the previous section, we have mentioned the various aspects of the game that determine the game experience. We mentioned the parameters that characterize the various artifacts involved in the gameplay.

All these parameters are listed in table 6.2, and associated with a range. The range for these parameters comes from game design requirements. They express that the light defense is less effective than the heavy one, that the sheep is slower than the eagle, etc. They determine what the game should be like, with regards to the artwork and story elements. From a game adaptation perspective, they are constraints that should be respected in order to keep the game fun and believable.

These ranges are expressed with the help of criticality functions (see section 4.2.3). This allows game designers to express more nuanced requirements on the ranges of the parameters, though we will stick to regular intervals in this example.

By observing the table, we can already observe that the **search space for these parameters is significantly large**. The number of situations we can put any player into can be calculated by multiplying the ranges of each parameter with each other. Even though the problem is continuous, we can pick a set of discrete values for each parameter, that we consider to be significantly different. In many cases, a small difference on the parameter value may not have an impact on the game experience. If we consider that each parameter has a set of 10 significantly different values, the search space is already as large as 10^{19}. This means that 10^{19} different configurations can be presented at any point to the player. This prevents any manual exploration to be effectively conducted.

Once all the relevant parameters are identified, the next step is to determine what makes a game experience relevant for a particular player. What are the things that we want the player to be able to achieve? What makes a game too easy, too hard, or well balanced? What are the objectives for the players, or, in other words, what are the relevant measures that we want the system to satisfy?

In the case of a tower defense game, there is a main measure that represents the success of a player on the game. The activity of the player consists in placing defenses in order to prevent enemies from crossing the map. The only goal of the player is to be able to actually

Table 6.2 — ASD-Tower defense parameters with their associated range, and their correlation with the percentage of defeated enemies (score)

			Range	Correlation w/ the score
Entity Type	Sheep	Speed	[30, 50]	-
		Robustness	[20, 40]	-
	Eagle	Speed	[50, 90]	-
		Robustness	[10, 30]	-
	Rhino	Speed	[10, 20]	-
		Robustness	[50, 90]	-
Defense Type	Regular	Range	[20, 40]	+
		Effectiveness	[10, 20]	+
		Pause between attacks	[300, 600]	-
	Heavy	Range	[5, 15]	+
		Effectiveness	[20, 30]	+
		Pause between attacks	[500, 1000]	-
	Light	Range	[40, 60]	+
		Effectiveness	[5, 10]	+
		Pause between attacks	[100, 400]	-
Wave Characteristics		Quantity of Sheep	[20, 40]	-
		Quantity of Eagle	[30, 50]	-
		Quantity of Rhino	[10, 30]	-
		Pause between spawns	[100, 3000]	+

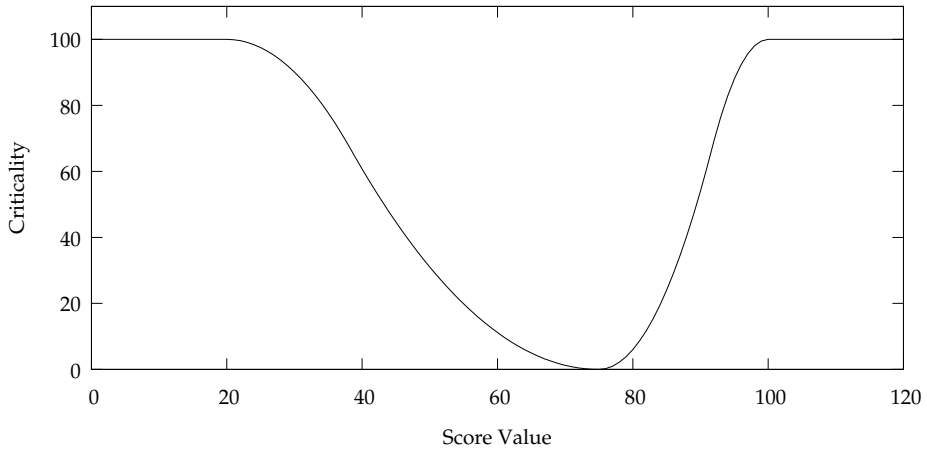
prevent these enemies from doing so. This measure reflects the players' activities, and how good these activities are performing on the game, and it is referred to as the *score*.

From a domain expertise, objectives need to be expressed on this measure. Such an objective is supposed to describe what the desired game experience is, neither too simple, nor to hard. Such an objective should also be expressed with the use of criticality functions. It is likely that in this case, the gradual capabilities of the criticality functions is relevant, since a *good* game experience is probably an approximate qualification. In our case, we suppose that a good game experience is a game experience where the player is able to reach the score of 75%. That means 3 enemies out of 4 are defeated by the player defenses. If the player is only able to defeat less enemy than that, we consider the game experience to be too hard, and if more than 75% of enemies are defeated, the game experience is considered to be too easy. This objective is defined with the criticality function shown in figure 6.16.

Of course, some game designers may argue that other values are better, but the main point of this experiment is to determine if the game adaptation system is able to modify the game experience to satisfy given requirements, not to determine what would be the best requirement for this particular game.

Now that the parameters, measures and objectives are identified, the game adaptation system can be instantiated. For each parameter, we create a parameter-agent, responsible for the modification of the parameter value. The modification is, of course, only allowed in the range defined by game designers (see table 6.2). For the percentage of defeated enemies, we

Figure 6.16 — Criticality function used to define the objective for the percentage of defeated enemies.



create a measure-agent, that will observe the game platform to obtain its value. The objective expressed by the designer, that characterizes a good game experience, is encapsulated into a satisfaction-agent, that will send requests to the measure-agent responsible of the score if necessary.

The final step before being able to use the game adaptation system is to list the correlation between measures and parameters. We want to list only monotonic correlations and, in this case, we observe that it is surprisingly easy to list them. Listing the correlations consists in determining which modifications on which parameters is likely to trigger a modification of the value of the measure. The measure here is the percentage of defeated enemies. Therefore, we can safely state that the strengths and speeds of enemies are values that, if increased, may reduce that percentage. Similarly, the strengths and ranges of defenses, if increased may increase the percentage of defeated enemies. Correlations involving the parameters of the waves are also straightforward. In a given wave, the number of enemies of each type, if increased, is likely to decrease to percentage of defeated enemies. And finally, the pause time before two entities spawn, if increased, is likely to increase the percentage of defeated enemies. The reason for that is that isolated entities are easier to defeat for defenses. Whereas if forming a dense group, one or several entities may receive all attacks while the others go untouched. These correlations are listed in the last column of table 6.2, and the measure-agent responsible of the score is made aware of them.

Given all these elements, the game adaptation system can be run along the game, in order for the game experience to be dynamically adjusted to the player, to better suit its needs. The next section details these experiments.

6.3.3 Experiments

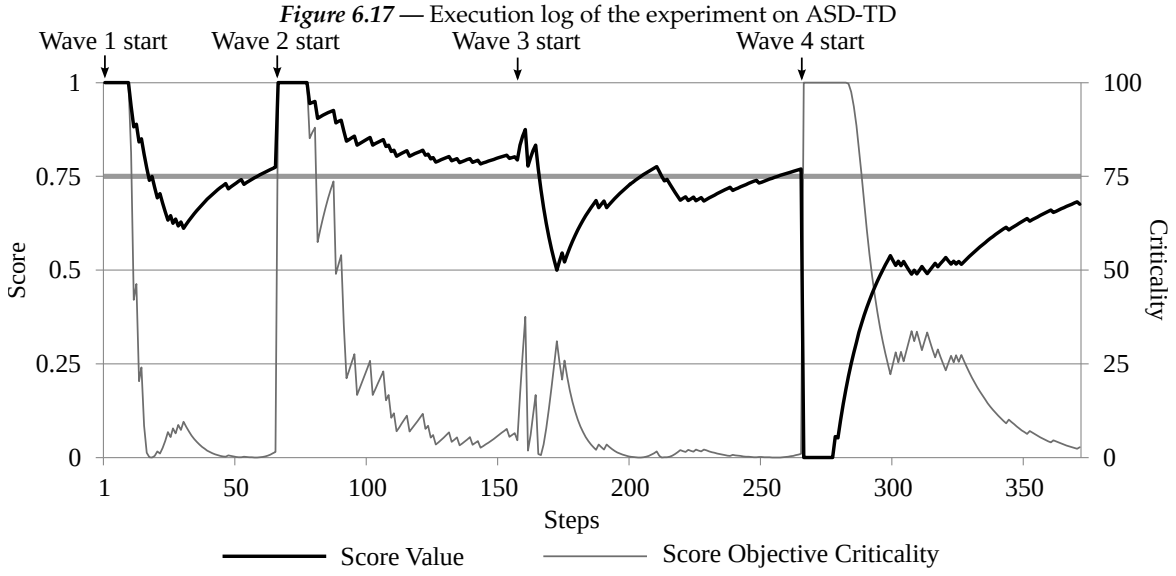
The purpose of these experiments is threefold. We first want to demonstrate the ability of the system to effectively adjust the game parameters so the objectives are satisfied. Then, we want to verify if the game adaptation system is able to deal with players that evolve during time, and get better as they play. And finally, we want to check if the system is able

to handle the fact that these progressions may not be linear, and that there are situations in which players may get less efficient, for whatever reasons.

Therefore, the player is here facing several waves. Between each wave, the player' strategy is modified, and more specifically, more defenses are placed, in a more efficient manner. Then, if the game adaptation system is able to handle these changes properly, the players' strategy is changed again, but this time, in a less efficient manner, by placing defenses in a suboptimal configuration.

The game ASD-TD is launched along with the multi-agent system responsible for the modification of the parameters in real-time, and we observe how the game adaptation system is able to react, by monitoring the value of the score at every points in time, as well as the criticality of the objective defined on the score.

Figure 6.17 — Execution log of the experiment on ASD-TD



The execution log of this experiment is shown in figure 6.17. The graph shows the evolution of the score during the game (black thick line) as well as the criticality of the objective defined on the score (grey thin line). The objective of 75 percent is also shown with the horizontal thick line.

When the game is started, the strategy adopted by the player is efficient. The player has defenses along the paths of enemies, and manages to defeat all of them. As a consequence, the score is 100 in the first few steps. According to the criticality function defining the objective for the score, a value of 100 yields a maximal criticality. Hence, the criticality curve is at its maximal level, denoting a completely unsatisfied objective. In other words, the game is, at this point, too easy for the player.

The non-null criticality value of the objective makes the satisfaction-agent send requests to the measure-agent responsible for the score value. This measure-agent then proceeds by sending requests to the parameter-agents. These parameter-agents react by modifying their value, and therefore, by modifying the values of the parameter of the game platform they are related to, which modifies the game experience itself.

As a consequence, the observed score starts to decrease around step 10, and goes beyond

the objective around step 25. Activity among agents is modified so the score can go back up, and by the end of the first wave, its value is close to the objective of 75%. As a consequence, the criticality curve is now null. The objective is satisfied and the challenge level is adapted to the player's strategy.

Before the second wave starts, the player's strategy is changed, by placing more defenses in more optimal positions. In other words, it is now more efficient, and is likely to win the game. When the second wave actually starts, the player once again defeats 100% of enemies. The same pattern is repeated. The score is maximal, the criticality of the objective on the score is also maximal, which triggers activity among agents, which eventually changes the values of the parameters.

Modifications of the parameters values change the game experience for the player, and the score is gradually modified as the wave is continued. By the end of the wave, around step 150, the score is close to the objective. The criticality of the score is near zero, and the player is neither in a state of frustration nor in a state of boredom.

A similar dynamics is repeated in the third wave. Before it starts, the player's strategy is enhanced: more defenses are placed in optimal locations. As a consequence, when the third wave actually starts, the score is higher than at the end of the second wave. Activity is generated in the multi-agent system, which eventually lowers the score. The score oscillates around the optimal value, and by the end of the wave, it is approximately on the objective, with a null criticality.

Before the fourth wave starts however, the player's strategy is changed in a different manner. Defenses are moved on the map so they are in less optimal places. In other words, the strategy the player adopts in this next wave is a worse strategy than previously. As a consequence, when the fourth wave actually starts, around step 170, the player is not able to defeat a single enemy, resulting in a 0% score, with maximal criticality.

In this case, the game is very far from being adapted to the player's skill level. The activity of the game adaptation system still manages to modify the game experience, resulting in an increase of the score around step 260. For the rest of the wave, the game is modified, and the score is increased to up to 70% at the end of the wave, with a relatively small criticality value.

6.3.4 Conclusion

We observed that the proposed approach **can be applied on actual games**, since ASD–TD is not a game designed to conduct experiments in computational intelligence, but it is an actual game, designed to be played for fun.

The proposed approach based on parameter modifications is well suited to this kind of game, as the game experience can be precisely and completely described with parameters associated with ranges, and objectives or requirements can easily be expressed with measures on the game.

The game adaptation system was able to dynamically modify various parameters in order for the game experience to satisfy arbitrary objectives, while keeping constraints sat-

isfied. The fact that the system adapts the game experience by modifying the parameters but always remains within the bounds defined by designers ensures that the designers ultimately have control over what happens in the game. They can create a good game experience, artistically speaking, and leave the complicated aspects of fine balancing the parameters to the game adaptation system.

We can also note that **the listing of the correlations was not problematic**. All the correlations have been intuitively found, and we did not have the case where a correlation was thought as being either positive or negative depending of the context.

We observed that the game experience was quickly modified to initially reach completely unsatisfied objectives. Moreover, we observed that when the player evolves, or changes strategy in any direction (for the better or the worse), the game experience is adapted accordingly to keep the objective satisfied, without making any assumptions about the future changes in the player's abilities, skill level, strategies and so on.

The next section presents another experiment on a different game.

6.4 Gameforge

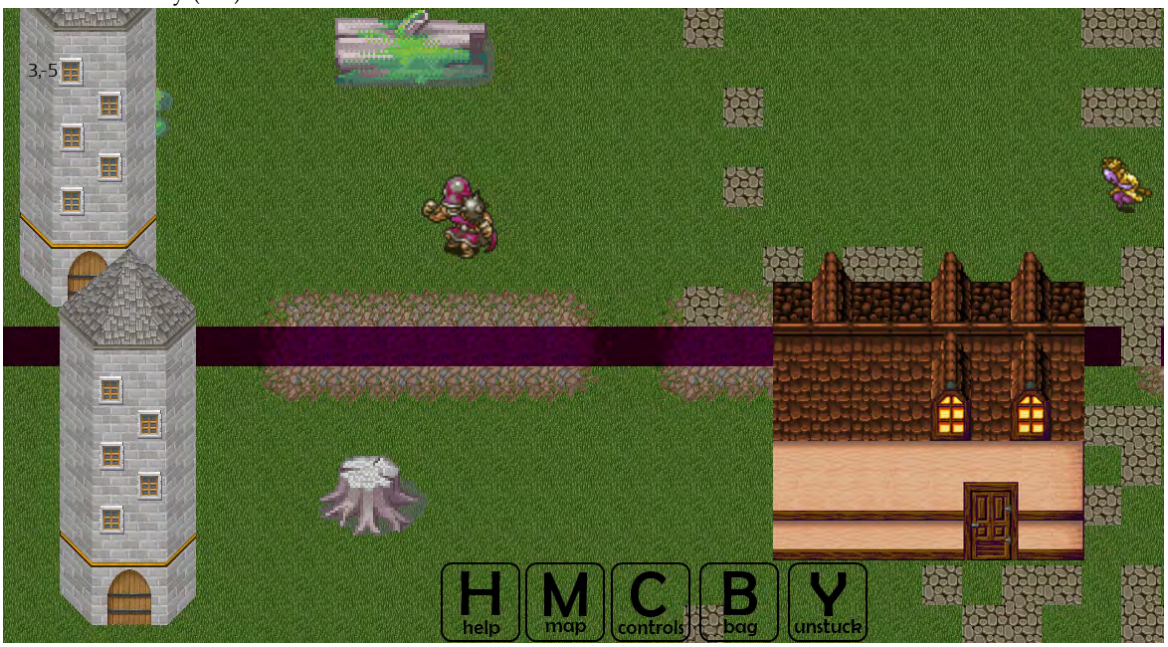
The *Entertaining Intelligence (EI) Lab*[2] in the Georgia Institute of Technology in Atlanta, USA, is interested in all techniques and approaches of computational intelligence applied to video games. They have conducted various experiments in order to modify different aspects of the game experience, from parameter adaptation, to storyline rewriting. They also proposed a game in which the topology of the environment in which the player evolves is generated based on preferences expressed by the player before the game starts. Some of their work is reviewed in chapter 2, see section 2.2.3 and 2.3.3.1.

During a three-month visit to this laboratory, our approach was applied to one of the game they were working on. This game is named *Gameforge*[3], and it is the archetype of classic adventure games. The player incarnates a character, evolves in a 2D environment, and can freely explore the map. As the player goes in specific spots on the map, a story is proposed, and the player can choose to follow that story, or wander randomly in the environment.

As the player travels in the environment, non-playing enemy characters are randomly spawned. If the player goes close to such an enemy, the game switches to a fighting phase. In the fighting phase, the player faces the enemy, and they alternatively attack each other. Each attack can cause the victim to loose a certain amount of health points. If a character (be it that of the player or not) reaches 0 health point, it dies. If the player dies, the game has to start over. When opponents are defeated, a small amount of health points is given back to the player's character. Figure 6.18 shows a screenshot of the player's character on the map at the extreme right of picture, facing an opponent at the center-left of the picture.

[2] https://research.cc.gatech.edu/inc/
[3] https://research.cc.gatech.edu/inc/game-forge/

Figure 6.18 — Screenshot of Gameforge map, with the player's character (right) and an enemy (left)



6.4.1 Applying the Game Adaptation System to Gameforge

Various parameters are used by the game platform to determine what the game experience is like. Such parameters first determine the maximal (and initial) amount of health points the player's character has, as well as the health points of enemies encountered on the map. The damages than can be inflicted to enemies by the player's character are randomly chosen in a range of values. Thus, two parameters are involved: the higher and lower bounds of that interval. Similarly, damages inflicted to players by opponents are randomly chosen in an interval specified by two parameters defining the lower and higher bounds. Another parameter represents the amount of health points restored to the player's character after each won fight.

Several things can be measured on the game while it is played, which can be used to characterize the game experience. Objectives should be expressed on these measures to specify what a well balanced game experience should be like for a given player. Two main measures are here considered. First, for each fight between the player's character and opponents, we can measure the number of rounds it took for the fight to be over. A round is a subdivision of the fight. A cycle of player's attack and opponent's attack consists of a single round. The number of rounds a fight may take is not limited. A second observation is the percentage of remaining health points the player has after a fight is won. This measure can be used to determine if the player has won easily, or if the competition was tight.

We arbitrarily define what a well balanced game experience is. We state that a fight should be won by the player in around 6 rounds. Less would be too easy, and more would be too long. Moreover, we state that the player should loose around 50% of his health during the fight. Less than that would not be challenging enough, and more than that would be too difficult. These two objectives are expressed with the criticality functions depicted in figure

6.19 and 6.20, so for any values the measures take, the objectives can continuously go from completely satisfied to completely unsatisfied. Again, some may argue that these values are not what a game experience should be like, and although we claim they are, that is not our point. We arbitrarily chose values to demonstrate the ability of our game adaptation system to balance the game according to predefined objectives. The relevance of these values is ultimately left to the appreciation of game designers.

Figure 6.19 — Criticality function defining the objective for the remaining health points after each fight

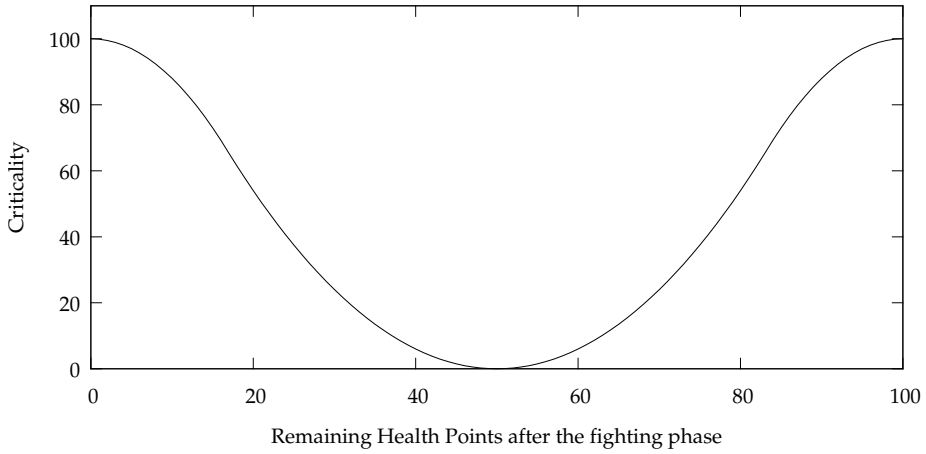
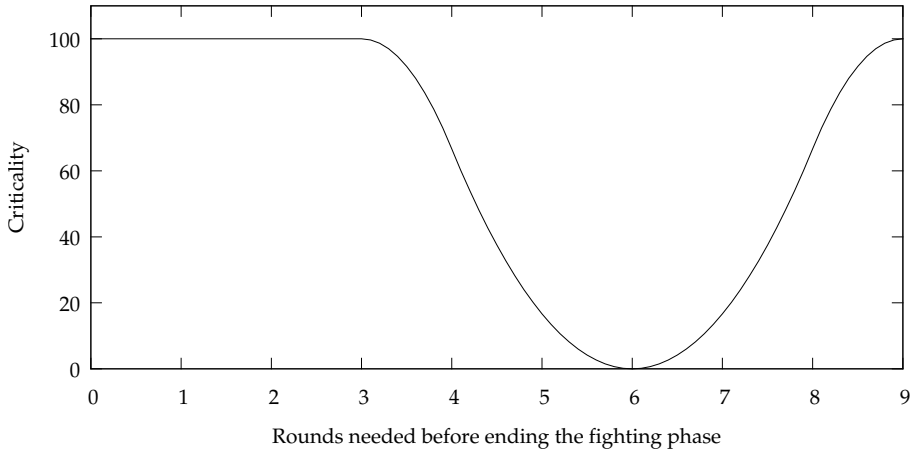


Figure 6.20 — Criticality function defining the objective for the round number for each fight



The direct correlations between these measures and the different parameters of the game engine are once again quite straightforward. The more powerful the player is, the less time (in rounds) it takes to defeat enemies. The stronger the enemies are, the less the player may have health points left after the fight.

Once all these elements are identified, the game adaptation system can be instantiated, quite similarly to previous applications. A parameter-agent is created for each parameter, a measure-agents is created for each measurable value, and a satisfaction-agent is created for

each criticality function. The game can be run along with the multi-agent system so it can be dynamically adapted to what happens during the game experience.

6.4.2 Experiments

The game itself was more designed by the EI lab as a tool for exploring various computational techniques, rather than an actual game designed to be truly enjoyed by players. Consequently, the player's abilities are not really taken into account to compute the outcome of the fights, as there is a single action available: the attack. The only viable strategy for a player in this game, is to attack opponents when fights are started.

To overcome this limitation, we chose to introduce a mechanism to simulate different skill levels for the player. We already mentioned that the amount of damage caused by the player's character to enemies is randomly chosen in an interval defined by two parameters. To simulate different skill levels, we propose to modify the probability distribution of that value. We propose three different skill levels. A *poorly* skilled player would have a greater probability of picking values in the lower third of the interval, whereas a *fairly* skilled player would have a higher probability of picking values in the middle third of the interval. Finally, a *highly* skilled player would have a higher probability of picking values in the last higher third of the interval. In other words, according to this simulated skill level, the damage inflicted by players would be different.

Thanks to this mechanism, experiences can be run to first determine if the objectives can actually be reached by modifying the parameters, and then if the change of the player' skill level can be handled by the game adaptation system.

These experiments have been run, and their execution log is shown in different figures. Figure 6.21 depicts the evolutions of the values of the two measures we have introduced: the player's remaining life (a) and the number of rounds fights take (b). Figure 6.22 shows the evolution of the global criticality, that is, at any point in time, the maximal criticality of all objectives defined on the game. Finally, to have an idea on how the parameters are modified, the evolution of one parameter, the enemies maximum health points, is shown in figure 6.23.

On these three figures, three distinct zones in time are delimited by vertical lines. These zones correspond to the three different skill levels of the players. At each vertical line, the skill level of the player is changed. It goes from *poor* to *fair* around step 140, and from *fair* to *high* around step 210.

At the beginning of the game session, (during the first 10 steps), we can observe that none of the objectives are satisfied. The first fight is over in 3 rounds (figure 6.21a), and the player ends up with around 90% of its health points (figure 6.21b). Consequently, the satisfaction-agents representing objectives on these measures are not satisfied, and start to send requests, which are ultimately transmitted to parameter-agents. As a consequence, the values of the parameters are changed, which has an impact on the two measured values. After 5 fights, we observe that the number of rounds has exceeded its objective (around step 50). The measure of the remaining health points after the fights reaches its objective of 50% after step 50.

Figure 6.21 — Evolution of the values of the two measures of Gameforge

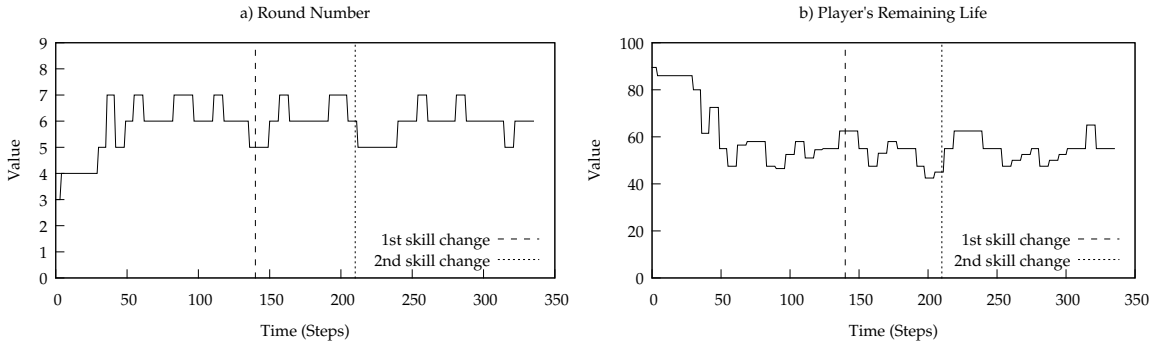
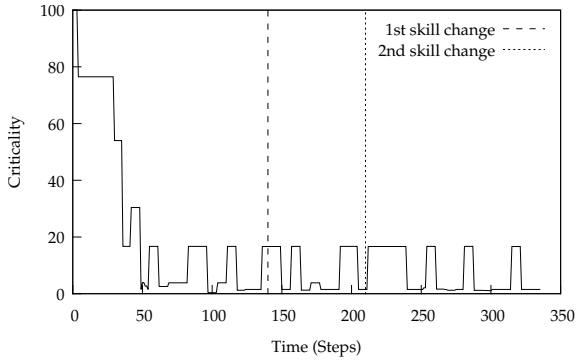


Figure 6.22 — Evolution of the global criticality



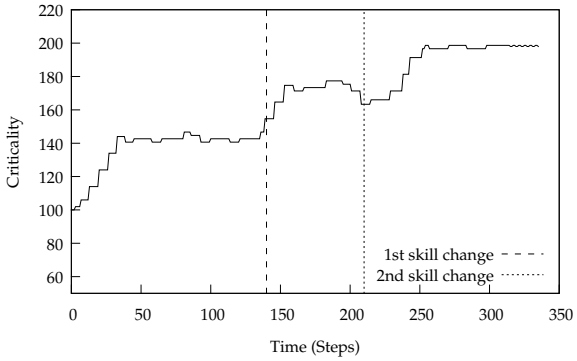
This dynamics can also be observed by looking at the global criticality chart (6.22). The global criticality is, at any point in time, the maximal criticality of all the objectives defined in the game session. This curve shows that the criticality starts to decrease at the beginning of the game, and remains quite low as afterwards.

Another way to look at the dynamics of the system is to look at the values of the parameters. For the sake of clarity, we only depict the evolution of a single parameter here: the enemies' health points (figure 6.23). It starts with a value of 100. At the beginning of the session, it quickly starts to increase, in order to help the two objectives to be satisfied. By the step 50, it is stabilized since the two objectives are satisfied.

By step 140, the simulated skill level of the player is changed, and set to *fair*, resulting in a probability distribution in favor of the values in the middle of the allowed interval to determine the damages caused by the player's character, making it more efficient against opponents. Interestingly, the measured values do not deviate from their objectives (figure 6.21), and therefore the criticality remains low ((figure 6.22). What happens is that the change in the skill level is quickly compensated by modifications of parameters, as shown by the example of the enemy health points parameter (figure 6.23), which quickly increases after step 140, before stabilizing around step 170.

The same dynamics are observed later, when around step 210, we change the simulated skill level again from *fair* to *high*. Again the measured values do not deviate much from their objective, as the skills change is compensated by the evolution of the parameters, as shows

Figure 6.23 — Evolution of the value of one parameter during the experiment: enemies' maximum health points



the example of the enemy health points, that is increased after the skill change before being stabilized around value 200 by step 260.

6.4.3 Conclusion

Similarly to ASD-TD, **Gameforge was not designed with this game adaptation system in mind**. The proposed paradigm, with the listing of parameters, the expression of objectives on measurable values, was once again adequate in this game. The game experience could be described in terms of parameters, as well as performances with measures. **Correlations were also easy to determine**, and there was no cases where a correlation was thought as positive or negative depending on the context.

The game adaptation system was able to dynamically modify the parameters of the game so that objectives defined on measures were satisfied. Even though no particular skills were involved, simulating different effectiveness of player's attacks allowed us to simulate various skill levels. We observed that evolutions of the abilities of the players were properly handled by the game adaptation system, by changing the values of the parameters to compensate these evolutions.

In the next section, we will go through another application, which handles real world requirements in the domain of maritime surveillance.

6.5 Serious Game for Real-World Applications

The work described in this section took place in the context of the Gambits[4] project funded by the French Government. One of the industrial partners is the French company DCNS[5], one of the world leader in naval defense. The activity of the company, in addition to ship manufacturing, involves the surveillance of territorial waters.

[4]Gambits (GAM-Based Intelligent Strategies) was a two-year project (2009-2011) funded by DGCIS (Direction Générale de la Competitivité, de l'Industrie et des Services) and approved by the PACA Marine Competitivity Center under the grant 09 2 93 0678. The other partners were Upetec (leader), DCNS and JFX.

[5]http://en.dcnsgroup.com/

In the context of **maritime surveillance**, operators need to monitor the traffic, in order to prevent various law infringements, such as illegal fishing or smuggling, but also other risks such as water pollution or even terrorism. In order to conduct these tasks, DNCS operators make use of a large panel of equipment to perceive the traffic. Such equipment is composed of coastal or patrolling radars, patrolling planes, radio contacts, etc.

These tasks require operators to process large quantities of data, coming from various sources. Particularities of the domain make it unlikely that any individual has a previous experiment in these tasks before he or she enters the company. For these reasons, DCNS put individuals in a training phase before they can actually monitor the traffic and make decisions about whether or not a particular ship needs to be intercepted.

In the training phase, individuals are placed in a simulated environment, where they can manipulate all the artifacts they need to master once they are in the real environment. The simulation offers them to visualize the traffic, and makes certain ships infringe the law, using various strategies. The complexity of the simulated environment as well as the complexity of the task demanded to players, make it difficult to propose an adequate experience. When different players are trained, they are using the same simulator, whether or not they have different skill levels or different abilities.

The goal of the project was to create a serious game intended to train individuals to the various tasks involved in maritime surveillance. Moreover, the idea was to include adaptive mechanisms in order to propose adequate game experience to players. They should be facing adapted challenges, so they do not get frustrated or bored. One of the main phases of the design of the game was to collect the domain knowledge from which the adaptation mechanisms can be built.

6.5.1 Domain Knowledge to Design an Adaptive Serious Game

To make the game adaptive, we divided the project into three parts described in figure 6.24.

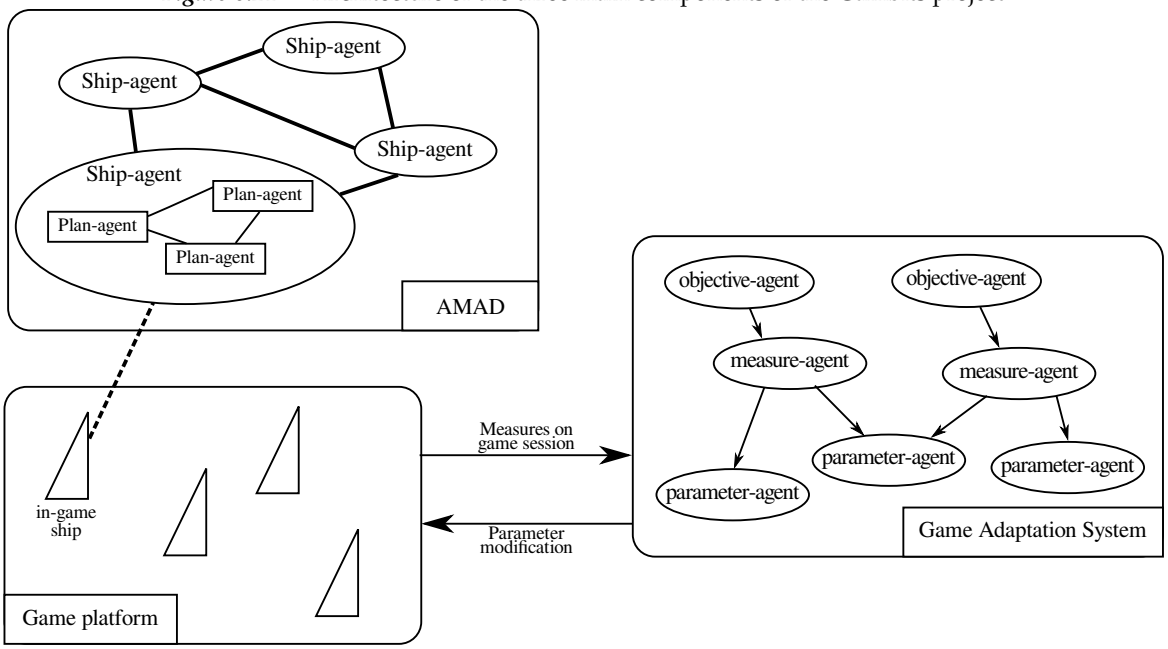
The game engine itself is constituted of ships evolving on a map, and reacting to various vector fields depending on their sensitivities. This game platform has been developped by JFX.

The behaviors of these ships is dictated by a set of plans. The plans used by the ships are determined by an external system named the Adaptive Multi-Agent Dynamics (AMAD) developed by Upetec. The plans can be created, deleted, or re-organized by the AMAD, in order for the ships to behave according to designers requirements.

Finally, the game adaptation system, described in this thesis, changes the parameters values of the game engine, in order to reach predefined objectives. We discussed with DCNS about which concepts are involved in the training of the individuals, and what are the possibilities to modify the game experience. DCNS offered a vast and complex description of the domain. The main concepts in the game which players need to be able to interact with are the various types of ships. Consequently, **DCNS provided a description of all the ships types that are typically detected by the company equipments** (see appendix C).

Ten ships categories were described. Each category, identified by its name (e.g., cargo,

Figure 6.24 — Architecture of the three main components of the Gambits project



yatch, fishing trawler, etc.), has several characteristics. First, it has a top speed, expressed in knots. Then it has a range for its Radar Cross Section (RCS), expressed in square meters, that determines the ease with which a radar may detect the ship (the bigger the ship, the larger the RCS, the easier it is for a radar to detect it): all the ships of a particular category typically have a RCS comprised in a given range.

In addition to the monitoring of the ships, players need to manipulate the various equipments available for their tasks. In different conditions, players may have different quantities of resources (or equipments), among which land radars. Each radar is defined by a range of actions, and has a probability of being defective. Players can also have a certain number of planes to inspect specific zones. Each plane has an specific autonomy (expressed in hours), as well as a visibility range, and a radar range. Ships patrols are available, and also have a visual and a radar range. These patrols have a lower speed, but have a greater autonomy.

Finally, the environment is characterized, since it has consequences on the game experiences of the players. Three aspects are most notably taken into account. The wind strength, expressed in knots, the height of the waves, expressed in meters, that has an influence on the detection means of the players, and the visibility, expressed in miles, which has a standard scale in the domain of maritime surveillance.

All these aspects are parameters that define the game experience. They all need to remain consistent, in order to keep the overall game experience believable, and realistic. Consequently, for each of these parameters, an authorized range is defined. Even when these parameters remain in these authorized ranges, the game experience can be greatly modified by changing the values of these various aspects.

An exhaustive list has been written by DNCS. This list can be found in one of the project reports, copied here in the appendix C. The amount of parameters as well as their range values describe a large search space. A quick calculation of the amount of different situations

in which a player can be gives us the number of 3×10^{23}. This number gives us an insight of the complexity of the domain, and strengthens the idea that a system able to dynamically modify the game experience would be a great advantage for the learning process offered by DCNS.

Domain experts then need to specify the desired characteristics of the game experience the player must have. They need to define the skill level that players should attain, as well as how the overall game experience should unfold. For that purposes, measures need to be introduced to effectively determine if a given game session is satisfying or not, according to the players' abilities, and to the domain requirements.

Such measures depend on what is actually implemented by the game platform, although certain measures can be thought of in advance. First, there is the quantity of infractions that a player is able to detect. Another measure is the duration of a game session. This requirement has been expressed by DCNS due to operational constraints, where training sessions must be timeboxed precisely. Though this requirement may seem a little odd and unrelated to what makes a *good* game experience, it still involves a measure on the game sessions, and therefore, nothing particular prevents us from considering it just like any other requirement.

From this domain knowledge, the instantiation of the multi-agent system to create game adaptation mechanisms is quite straightforward. For all the identified parameters, a parameter-agent needs to be created, and made aware of the authorized values for this specific parameter. For all measures expressed by domain experts, measure-agents are created. They are responsible for observing the game platform to keep their values up to date. Finally, for each requirement expressed on the game experience, a satisfaction-agent is created, with the appropriate criticality function representing the desired values for that specific requirement.

Finally, we need to list the correlations between the measurable values and the parameters. Again, these correlations do not consist of a model of the game platform, since we do not consider weights, but only signs. We observed that the correlations were quite easy to determine based on the knowledge coming from the domain and with the use of common sense: the amount of detected infractions should be increased if the quantity and the quality of the means of observations of the players are increased. Similarly, higher speeds from ships may allow them to escape from the sight of the player, and a low RCS may help them remain undetected. The correlations with the environmental parameters is also easy to determine. High waves, low visibility and strong wind make the observations means less efficient, and thus, may allow illegal ships to operate undiscovered.

The game session duration – the time it takes for all the ships in a given scenario to go from their starting point to their destination – depends on various parameters. Some correlations can be listed: the higher the speed of the ships, the quicker the game will end. The less the environmental conditions slow the ships down, the quicker the game will be able to end. Regarding the populations of ships, the more ships are supposed to be spawned in a given game session, the more time this session will take. Detection capabilities of the players do not have an obvious correlation with the time a given session may take.

Of course, the proportion in which these various parameters influence the measures are unknown, and probably depend on the context and/or the abilities of the players, but we

just need the sign of the correlation at this point. Each measure-agent is made aware of which agents it is correlated to.

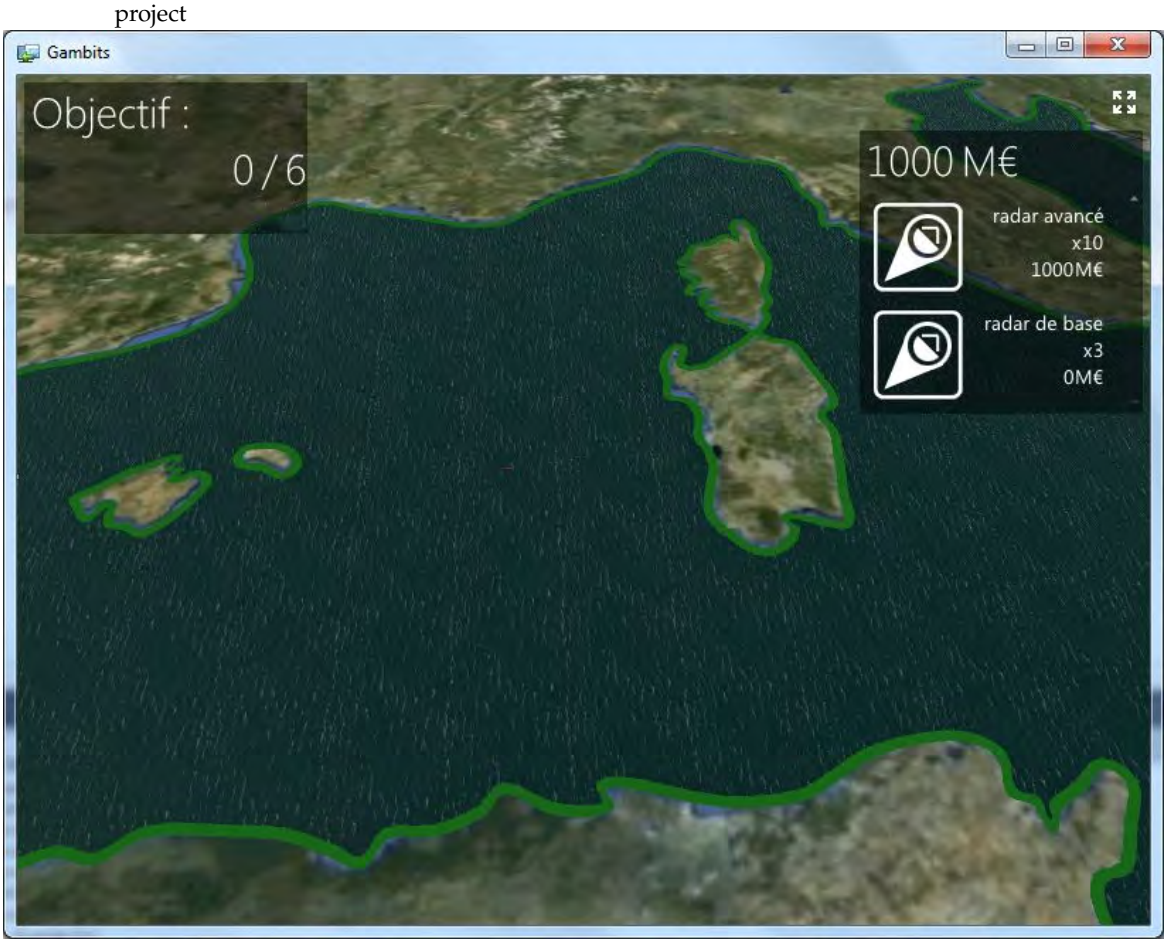
Once this information is gathered, and the agents are instantiated, the game session can be launched along with the game adaptation system.

6.5.2 Experiments with the Serious Game

Several experiences can be conducted using the game platform developed in the context of the project. Unfortunately, by the end of the time period dedicated to the project, the progress made by the development team on the game platform did not allowed us to make advanced experiences with operators from DCNS, due to the lack of actual game features in the platform: most parameters were not implemented in the game engine.

The platform features the modeling of the Mediterranean sea, as well as the navigation of a large quantity of ships (see screenshot in figure 6.25). In order for the ships to be placed on the map, routes need to be defined, going from one starting point to a destination. Then, ships are added to the route, and travel at their defined pace on this route. While they sail on the route, the ships try to avoid every obstacle that cross their paths, be it another ship or the shore. A parameter defines how much ships are repulsed by any obstacle.

Figure 6.25 — Screenshot of the game engine developed in the context of the Gambits project



The game also takes the wind into account. The wind is set in a particular direction, with a given force, and the actual speed of a ship takes this wind into account. In other words, the wind has the ability to slow the ships down.

The user of the game has the possibility to place radars on the land. These radars have a preset range, and each time a ship goes through the field of the radar, it becomes marked.

We then introduce two measures on the game platform. In order to stay consistent with the nature of the requirements coming from the domain, we choose to measure the quantity of ships detected by radars placed on the shore, and secondly, the time it takes for all the ships to finish their route. Even though these measures do not reflect exactly what was described by DCNS, they are somewhat of the same nature. Similarly, even though these mechanisms do not reflect the actual complexity of the domain or the tasks demanded to DCNS operators, they still have allowed us to conduct experiments to observe how the proposed game adaptation system behaved.

Before running some experiment, we launched the game with no adaptation mechanisms at all. For all the parameters present in the game platform, we choose values in an uninformed manner, and launch the game. We observe ships going through their routes, and after around 4 minutes and 30 seconds, the game session is over; all the ships have reached their destination.

The remaining of this section describes three experiments done with the game adaptation system applied to the same simulation. We first dynamically modify the game experience in order to satisfy one objective. We then try to adapt the game experience so a different objective can be satisfied, and finally, the last experiment combines the two objectives at the same time.

One objective on the duration

The objective chosen for the first experiment expressed the fact that the duration of the session should be of 3 minutes. Consequently, a satisfaction-agent is created, sending requests to the measure-agent representing the duration if necessary. After launching the game, once the first ships reach their destination, their average time, as well as the total number of ships in the wave, are used to estimate the total duration of the game session.

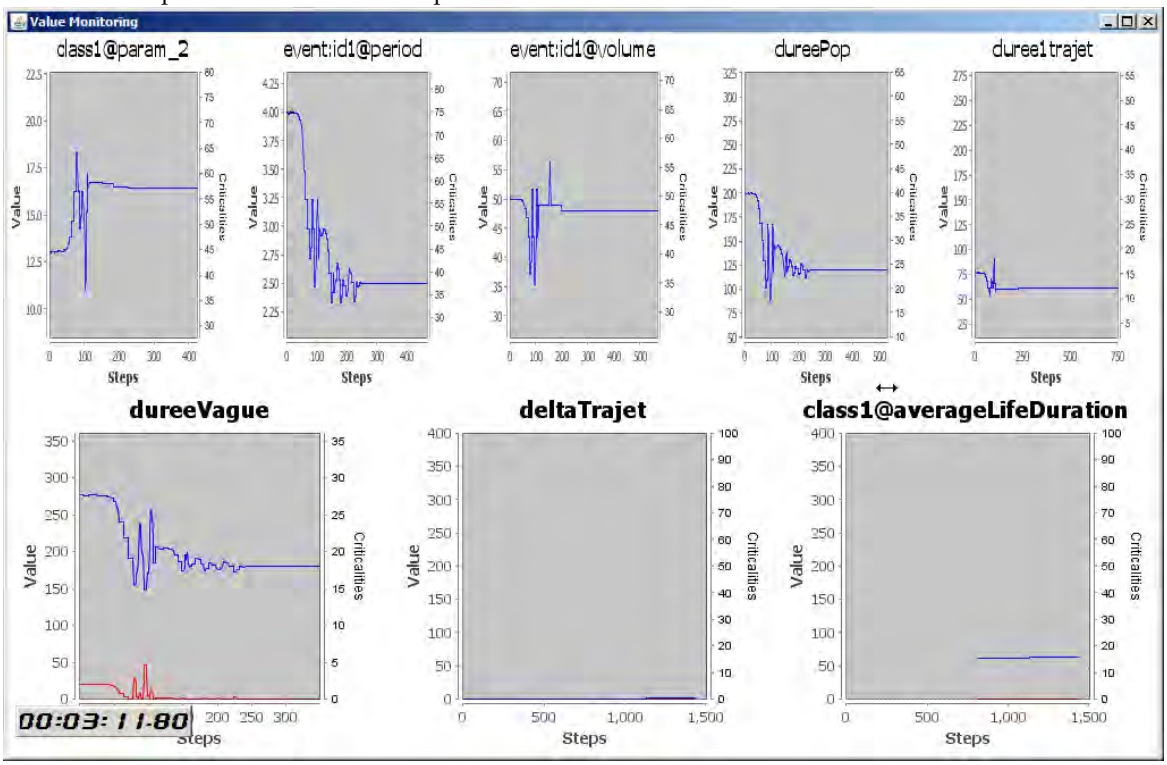
If this estimation is different from the 3-minute objective, then the satisfaction-agent that encapsulates the 3-minute objective computes a non-null criticality. It therefore sends requests to the measure-agent responsible for the duration of the game session. This measure-agent is correlated to several parameters (such as the ships speed or wind strength), consequently it sends them requests for value changes.

The activity of the parameter-agents changes the values of the parameters, notably the speeds of the ships, as well as the strength of the wind, resulting in a general speed up of the game session. As the parameters values are modified, other ships reach their destination with a different timing. This enables to update the total estimation of the duration of the session and consequently, to generate more activity in the system, in order to keep adjusting the parameters. These parameter modifications make the game session shorter, so it finally ends in 3 minutes.

Figure 6.26 is a screenshot of a video demonstration we produced during the project,

where several measures and parameters are monitored. Among the several charts depicted on the figure, the one on the bottom left with the title "dureeVague" shows the evolution of the wave duration estimation expressed in seconds. We can observe that the estimation starts too high, around 275, and is progressively modified to finally being stabilized around 180 (3 minutes) by the end of the session. In the meantime, we can observe the variations of various parameters on the first row in the picture, such as the speed of ships (class1@param_2), the time between each ship creation (even:id@period), and the total amount of ships in the wave (event:id1@volume).

Figure 6.26 — Screenshot of a demonstration video showing the monitoring of measures and parameters for the first experiment



One objective on the detection rate

For the second experiment, we remove the previous objective, and instead specify another requirement, involving the amount of ships detected by radars placed on the shore. We consider that a game experience is adequate if a large part of the sailing ships is detected, but not all of them. Hence, we introduce an objective of 80% of ships detected by means of observation, i.e. radars.

The game is launched, and a radar is placed in a specific position on the shore. At the beginning of the game, as the first ships navigate from their starting points to their destinations, all of them are detected. The measure-agent responsible of the percentage of detected ships calculates the percentage so far, which turns out to be 100%. The satisfaction-agent that encapsulates the objective of 80% observes this measure-agent, and sees that its current value is above the objective. Consequently requests are being sent to parameter-agents (such as visibility of the environment, or range of radars). These parameter-agents start

to modify their values, which then modifies the game experience: newly created ships are more furtive, the environment is changing, detection means are less efficient, and the ships are more spread out. Therefore some of the chips start to manage to reach their destination without being detected, and as a consequence the score observed by the measure-agent starts to lower . By the end of the game session, around 80% of the ships have been detected in approximately 4 minutes and 30 seconds.

Two objectives combined

The final step in the series of experiments conducted with this game engine consists in defining two requirements at the same time. In other words, two satisfaction-agents are created at once, one expressing the fact that the game session should last 3 minutes, and the other that expresses that 80% of the ships should be detected by radars.

The same initial conditions as before are set, and the game is launched. A radar is placed in the same spot as in the previous experience. Similarly to the second experience, since the first ships are detected by the radar, a rate of 100% is calculated, resulting in more furtive ships, and harder detection conditions. As the first ships reach their destination, an estimation of the duration of the game session is calculated (over 4 minutes at that point), resulting in changes in the parameters values in order to shorten the game session. In parallel, the rate of detected ships is maximal, resulting as well in an activity among the agents in order to reduce this rate.

The interesting fact is that we now have two overlapping objectives, and some parameters are involved in both. T**herefore, an equilibrium needs to be found in order for the two objectives to be satisfied at once**. Parameters involved in only one objective will modify themselves a bit more so that parameters involved in the two objectives can remain at a value where they can satisfy both. Without any planning in advance, proper values are found, and the two objectives end up satisfied.

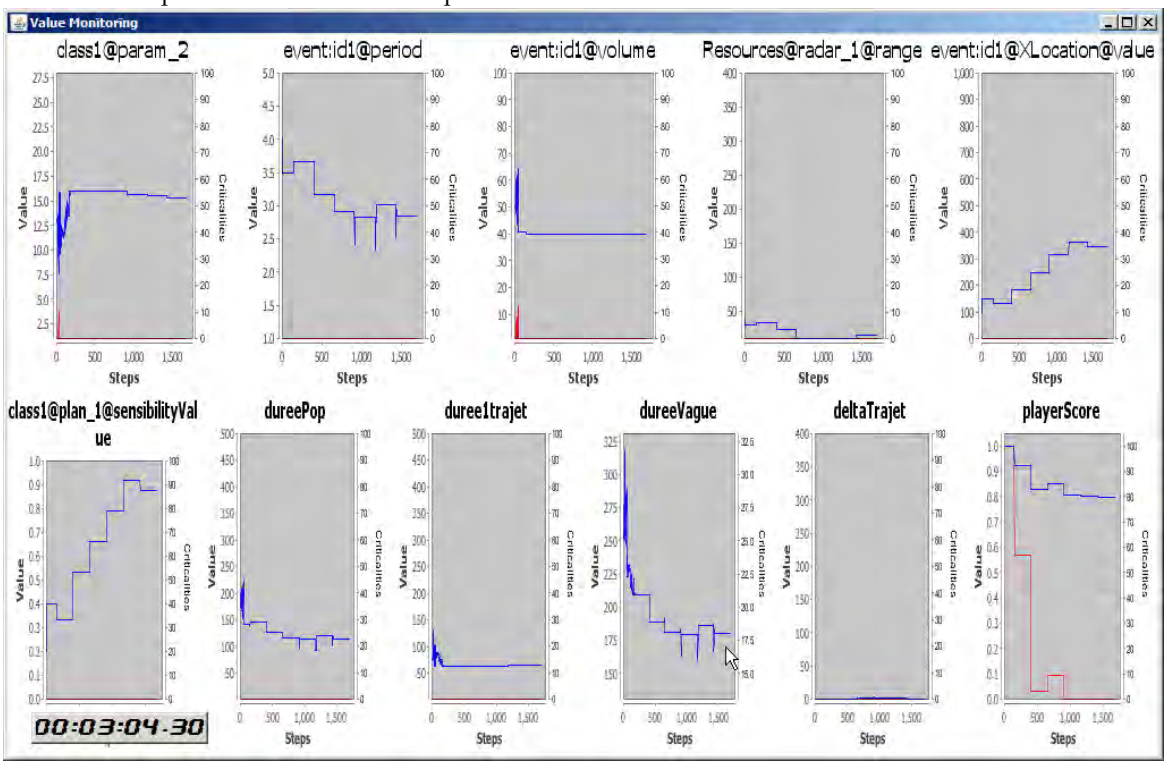
Similarly to the first experiment, figure 6.27 shows a screenshot of a video made for demonstration purposes, showing the monitoring of various measures and parameters on the experiment. The chart titled "dureeVague" shows the evolution of the estimation of the duration of the wave, which starts around 275 (like in the first experiment) and is gradually decreased to 180 (3 minutes). In parallel, the bottom right chart titled "playerScore" shows the detection percentage of the ships (here expressed as a value between 0 and 1). It starts at 1, and is gradually decreased to finally be stabilized around 0.8. We can note that the activity of the parameters due to the objective related to the score generates activity that affects the value of the duration; the "dureeVague" curve shows more variation than in the previous experiments, but these are quickly compensated, and the value remains stable overall.

6.5.3 Conclusion

This application of the game adaptation system differs from all the others, in the sense that this time, domain experts are actually requiring mechanisms that would allow a game to adjust itself, according to what human players are currently doing with it.

By exchanging with experts from the field of maritime surveillance, we were able to discuss their requirements in terms of gameplay, and difficulty characterization. The de-

Figure 6.27 — Screenshot of a demonstration video showing the monitoring of measures and parameters for the third experiment



scription of the domain is quite complex, as the tasks of the operators to be trained feature many different aspects at the same time. The conducted work is encouraging, since **we were able to express the knowledge of domain experts with the paradigm we propose**, namely, with parameters, measures and objectives.

Once again, we observed that **correlations between the various elements of the scenario did not cause troubles**. Correlations were listed, and **none of them were ambiguous regarding their sign**.

The game platform itself was not completely satisfying, as it lacked features to demonstrate the full applicability to the domain. We managed nonetheless to model maritime surveillance problems with the concepts present in the game platform, and successfully run the game adaptation system to satisfy the objectives defined on the measures, while keeping the different parameters in the ranges they were assigned to.

Additional work, mostly new features from the game platform, is needed to claim that this game could improve the training process of DCNS operators, but the approach has satisfied domain experts, that see in these adaptation mechanisms an opportunity to tailor situations to trainees' skills and abilities without having to go through undesirable one-size-fits-all scenarios for all their operators.

7 Conclusion and Perspectives

As a conclusion, the contribution of this work can be viewed from different angles. It can be seen as a solution to the problem initially described in the introduction of this thesis. **Games need to be dynamically adapted to be efficient**, whether for learning or for fun. But another way to look at it, is by considering its context.

By making use of the AMAS theory, we ran into challenges that are not specific to our application, and therefore, made **contributions that can potentially be reused by other works**.

Finally, these two angles can also be considered when imagining potential evolutions of the work. Room for improvement exists with an applicative perspective, but other useful contributions in a more abstract fashion can already be thought of.

The following sections summarize the contributions of the thesis at these two levels, and discusses about several improvements.

7.1 Applicative Contributions

The increasing use of video games for both entertaining purposes and pedagogical purposes has created new challenges in terms of game design. Studies have shown that **game experiences should match abilities of players in order to be efficient**, both in terms of entertainment and in terms of learning. The ever growing population of players have significantly increased the diversity of individuals, and consequently, the diversity of abilities and skill levels, making the design of one game that may fit any individual needs nearly impossible.

For these reasons, means for adapting the game experience to players have started to gain attention from several communities, both academic and industrial. We have proposed **a system able to manipulate game experiences in real-time**, in order to satisfy predefined objectives. Given the various constraints and conditions under which adaptation mechanisms should be able to work, there are several characteristics that we believe to be necessary for such mechanisms to be completely satisfying. We have considered them all along this thesis in order to meet these requirements.

- **Genericity**: there is a **significantly large diversity of games** created each year. All these games may use different mechanisms, and feature various concepts in their gameplay. For a game adaptation system to be able to function in a majority of these games, **it**

should not rely too heavily on specific game concepts or artifacts. It should remain abstract enough in order to be instantiated when a given game needs adaptation. **We have based our system on general concepts: parameters, measures and objectives**. We claim that these concepts can be found in a large majority of games to represent more specific mechanisms and artifacts. We have successfully used them in a simple simulation with a prey-predator model, as well as in two different games, with different game concepts. Moreover, we have successfully described the requirements and various game aspects of a real life business domain, by exchanging with experts of the maritime surveillance domain. All these experiences are significantly different, and all of them functioned well with the proposed paradigm of measures, objectives and parameters.

- **Ease of instantiation**: people that need such adaptation mechanisms are generally not specialists of computational intelligence. An efficient tool to adapt game experience should be **understandable and manipulable by those who actually need it**, i.e. game designers, or experts from a particular discipline, in the case of learning games. If proposed mechanisms require a high degree of expertise, their benefit might be canceled by their cost of instantiation. In addition to be applicable to a large variety of games, the use of measures, **parameters and objectives makes it really easy for domain experts or game designers to apply the approac**h. These elements can be used to describe games with minimal efforts, since **they represent actual game concepts, and are usually already used by designers and pedagogical experts**. No particular knowledge or expertise is required to describe games and objectives in the paradigm introduced in this thesis, as observed when it was applied to the domain of maritime surveillance.

- **Complexity handling**: the complexity of modern games is constantly increasing. More and more rules are involved in the gameplay, which features more and more different entities, with their own properties. The environment of games becomes larger and larger, and the means of interactions of modern platforms bring even more possibilities. A system designed to adapt a game experience to players **should be able to manipulate games with a great complexity**. Consequently, no exhaustive search, complete modeling of game concepts or analytical approaches on game mechanisms should be necessary. These may work on simple games, but we are interested in actual, complex games. As the examples of generated problems demonstrated, **we do not need to have an accurate model of the system we are trying to control**. Basic information like correlation is enough to modify the parameters so that objectives on measures are satisfied. **No predictive abilities are necessary, nor knowledge about how much a parameter is involved in the satisfaction of an objective**.

- **No assumptions about human players**: Human populations feature many different individuals. People all have their own way of dealing with the challenges presented in games, their own skill levels and progression abilities on different aspects on the gameplay. Some are constantly struggling to find different strategies, or unforeseen way of tackling the game challenges. Consequently, we argue that **analyzing players, or trying to predict their actions is bound to fail**. Instead of trying to predict what players will do or want, we claim that a satisfying system should react to their actions,

without making any strong assumptions about them. In the proposed approach, the game adaptation system reacts to what is observed in the game. When measured values change, and objectives are not satisfied, parameters are adjusted in order to modify the measured values towards their objectives. **Nothing is *learned* about players**, and therefore, no assumptions are made about the way they may think, plan their actions, or react when facing certain events. Consequently, whatever the reactions players may have, whatever their way of handling particular game events, the game adaptation system is not penalized. **No information about the player is required, nor constructed as the game experience is modified**.

- **Adaptivity**: finally, the coupling of complex games and changing players makes the system to adapt highly dynamic. **What works at one point in time may not later on**. Players may change in unexpected manners, and a slight modification of the game experience may have significant consequences, depending on the context. These nonlinearities in the system composed of the player and the game platform are challenging, and game adaptation mechanisms should be able to handle them. As mentioned in the previous point, the proposed system only reacts to what is observed, and does not plan actions in advance. Consequently, even if **the nature of the game or that of the player significantly changes over time**, the adaptation system will not be penalized, since nothing was actually learned. Changes in the game or in the player will just provoke new parameter changes. If an action from the adaptation system had a specific consequence at a given point in time, this consequence is not remembered. **The system does not assume that the same consequence will happen if the action is applied later**, even if the context seems similar. As the prey-predator model example shown, the system is able to deal with significant dynamics. When the player changes as well, the system is able to adjust its strategy to still satisfy the objectives, as demonstrated in the two games, where the player changed over time.

These several aspects make us think that the proposed expression framework constituted by parameters, measures and objectives, along with the multi-agent system responsible for the modification of the parameters is well suited for the growing requirement of adaptation of game experiences to players, in both entertainment games and pedagogical games. It is able to easily work with modern and complex games, and to be used by non specialists, with a large variety of players, changing over time.

7.2 Scientific Contribution

This work also constitutes a contribution in a field broader than the video-game adaptation. The fact that we considered parameters, correlated with measures, and objectives to satisfy on these measures, allows us to imagine **applications outside the scope of video games**. Many domains can be modeled with inputs correlated to outputs. Consequently, this approach could be applied in the field of control. Control engineering is interested in a wide variety of systems, such as manufacturing systems, bio-processes, and so on. In the field of control, target systems are often expressed with a formalism similar to the one we have used

all along this thesis. The approach could therefore be applied in many cases where inputs need to be modified to satisfy objectives on outputs.

In the AMAS theory, systems are designed in a bottom-up approach. The constituting parts of a system are considered individually, as opposed to the whole they form. The theory states that to be functionally adequate, the whole needs to have cooperative interactions with its environment. Two systems having cooperative interactions means that their interactions reinforce their activity in a positive way, rather than being neutral, or even preventing their activity to properly occur.

For a system as a whole to have cooperative interactions with its environment, when designing an AMAS, the theory states that all its constituting parts should also have cooperative interactions. All these constituting parts, the agents, have a nominal behavior, that can be seen as the normal case, where no particular problem occurs. Due to changes in the environment of the system as a whole, interactions between agents may become antinomic. In such cases, agents will try to solve the situation, and have their interactions to go back in a cooperative state. For that purpose, one of the mechanisms introduced by the theory is to tune values that can be modified by the agents.

Work has already been to apply the AMAS theory to system control, as detailed in section 2.4.4. In this approach, control actions in certain contexts are learned in order to obtain an optimal control policy. Though this work has been successfully applied to several application domains, we chose to take a slightly different approach. We believe that in order to control systems like video games, the lack of memory is an advantage, since these systems exhibit a significantly large amount of possible situations, which may never be encountered more than once. Consequently, an approach based on the learning of contexts may not be suitable.

By contrast, the strength of the approach we chose consists in the fact that **the control system is reactive, and does not need to learn a model of the system to be controlled**. Highly dynamic systems can therefore be controlled by the proposed multi-agent system, granted that enough observations can be made on the outputs. The fact that the control system is purely reactive has allowed us to **consider systems in constant interactions with humans**. Usual control techniques often need to be applied on deterministic, linear problems. Our ability to quickly handle dynamics, without being penalized by previous experiences is a great advantage to deal with nonlinear dynamic systems, where traditional techniques may need additional time to perform adequately again.

Therefore depending on the nature of the system one wants to control, **one can choose between an approach based on context-learning, or a memory-free adaptive approach more similar to ours**, where no learning occur.

In the approach we have proposed, we gave a great importance to parameter tuning, since it represents the main activity of the system. However, in many other systems based on the AMAS, completely unrelated to the adaptation of game experience, parameter tuning is also of a great importance in the functionality of the system. Agents often tune their values when receiving requests from their environment. For that purpose, the Adaptive Value Tracker (AVT) has been developed in the context of the theory, and we have used it to build our system (see appendix B). The AVT provides an adequate strategy to tune a value

based on successive *greater* or *lower* inputs. The value of the AVT is modified either fast or accurately depending on how close the AVT believes it is to the optimal value.

One of the problem we have encountered when designing our system is the fact that **the information used by agents to tune their value was not necessary relevant information**. Indeed, information perceived by agents may not reflect the reality **because of delays that may exist between agents**.

Various reasons may explain these delays. For instance, **communication delays**: legitimate information is sent from an agent to another, but by the time this information reaches the recipient, it is no longer relevant. Other delays are more inherent to the problem one is trying to solve. Agents may tune a value, but the consequences of that tuning **may not be immediately effective**, from the point of view of other agents. As a consequence, these latter may consider that additional modification is needed, even though it is not, and they just need to wait a moment before they actually observe the consequences of the tuning.

Not taking these kinds of delays into account may result in **oscillations around optimal values**, and may prevent the agents to re-establish cooperative interactions, and thus, prevents the system to be functionally adequate.

In this thesis, we have proposed **mechanisms for agents to be able to take these delays into account**. We have described behaviors with which agents constantly estimate the delay that may exist between them and other agents they interact with. These delays estimations allow them to make better decisions, and ultimately, to prevent oscillations around optimal values, so they can maintain cooperative interactions with their environment, and therefore let emerge a functionally adequate behavior of the system as a whole.

The **delays handling mechanisms are designed independently from the application domain**, and therefore, they can be used as is, provided that agents modify values based on information from their environment, in a context where delays may exist, which is often the case when working with the AMAS theory.

7.3 Applicative Perspectives

An interesting perspective, that can be imagined from this work, is **to use the proposed paradigm and system to balance games**. A balanced game is a game with parameter values that satisfy a set of requirements expressed by designers or experts. As opposed to what we claimed in this thesis, designers often try to create *one-size-fits-all* games.

Since all players are different, and since players learn different things at different paces, a single game configuration for all the player population may result in a sub-optimal game experience for the majority of players. However, due to various constraints, dynamic parameter adjustment is sometimes refused. In these cases, compromises need to be found to decide what the parameters values should be.

The usual process to determine parameters values is to have a population of testers. These players play the game intensively, and the designers observe their game sessions. When a problem is observed, designers use intuitive knowledge to manually change the value of some parameters. The testing population then plays the game again, and designers

observe if the problem was solved. This cycle can be repeated many times before designers are satisfied, or before project constraints force the team to make a final decision [Bethke, 2003].

Instead of manually tuning the parameters, the system we have proposed in this thesis could be used to find better values. The difference with the experiences we conducted, is that all the changes made by the system on any parameters should, somehow, be retroactive. This means that a parameter configuration should be tested in a large variety of situations with a large variety of players.

One idea is to keep these testing players, and to make all of them play with the same parameters. When an objective is not satisfied by one of the player, the adaptation system changes the parameters, and the new values are applied to all the players, in order to detect potential new problems. There could be a significant gain of time compared to manual tuning.

Another idea would be to have **means of automatically exploring players' strategies**. Such mechanisms could be used to find players' strategies that do not satisfy objectives, and the adaptation system could consequently modify the parameters to increase the satisfaction of objectives, resulting in a full automation of the balancing process of a game. Of course, automatically exploring players' strategy may not be possible depending on the game type, especially those with a large search space.

These considerations lead us to **the problem of multi-player games**. From our point of view, nothing prevents us to dynamically change the parameters in order for objectives to be satisfied in a multi-player context. Measures on multi-player contexts could easily represent equilibrium between players with different strategies or skill levels. However, some may still refuse that games could be modified in real time. In this case, the same ideas still stand. We need means to explore players' strategies to automatically tune parameters in order to satisfy high-level objectives, be it automatic exploration, or with a testing player population.

Whatever the context, we firmly believe that the developed approach could be of a great help for professional game designers' tasks and challenges.

7.4 Scientific Perspectives

When considering not only the video game domain, but a broader scope, we can imagine other improvements of the proposed system.

One aspect of the described system we are currently interested in is **the correlation matrix**. The correlation matrix is used to list correlations between measured values on the game – or more generally on the target system – and the parameters. A correlation may exist between a measure and a parameter, and if so, it is characterized by a sign: either positive or negative. If positive, it means that a positive variation of the parameter is likely to trigger a positive variation of the measure. If the correlation is negative, it means that a positive variation of the parameter is likely to trigger a negative (i.e. reversed) variation of the measure.

In the correlation matrix, there are no information about the weights of correlations, but only signs. Moreover, there are no contexts. This means that experts only list direct

correlations, in other words, those that are obvious to them, and not the correlations that *may* exist if we consider side effects. The actual influence of parameters on measures, that may depend on the context, as well as potential side effects are handled by the multi-agent system.

Due to the nature of the information given by experts, only monotonic correlations can be listed. In the experiences described in chapter 6, we have observed that it was not a problem, and that in all the cases we have studied, correlations were actually monotonic. But despite that applicative consideration, it is interesting to investigate cases were non-monotonic correlations exist.

If a non-monotonic correlation exists between a parameter and a measure, this means that in some contexts the correlation may be positive, and in some other contexts, it may be negative. In the current state of the approach, this correlation could not be listed in the matrix, and consequently, could not be used by the adaptation system.

The idea we propose to explore is to give the adaptation system **means to learn these correlations**. If correlations between parameters and measures could be learned, then non-monotonic correlations could be determined depending on the context in which the system is. If the context changed, and consequently a correlation went from positive to negative, **learning abilities could make the system update the correlation**, to keep using it with its new sign.

Correlation learning abilities could also help to find correlations that experts did not think of, or potentially to correct them if they made a mistake.

One question about these learning capabilities, is to determine if we are able to learn changing correlations faster than they change. If we are able to do so, then it would be a great improvement of the system.

In fact, work has already be done in this direction, and we currently have promising results [Jambu, 2013]. We consider that the modification of an observable, is the result of the weighted sum of the modification of the parameters. By observing the modifications on the inputs and the outputs, we try to learn the weights. If these weights – positive or negative – can be determined, then the correlation can be deduced from them.

Additional work needs to be done to determine how these techniques can be applied in our case, but it is nonetheless a promising idea.

Bibliography

Aamodt, A. and Plaza, E. (1994). Case-based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Commun.*, 7(1):39–59.

Abt, C. C. (1970). *Serious Games*. Viking Press.

Alvarez, J., Rampnoux, O., Jessel, J.-P., and Methel, G. (2007). Serious Game: Just a Question of Posture ? In *Artificial and Ambient Intelligence convention (Artificial Societies for Ambient Intelligence) (AISB (ASAMi)), Newcastle upon Tyne, UK, 02-04/04/2007*, pages 420–426, http://www.aisb.org.uk. The Society for the Study of Artificial Intelligence and Simulation of Behaviour.

Amato, C. and Shani, G. (2010). High-level Reinforcement Learning in Strategy Games. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, pages 75–82, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Andrade, G., Santana, H., Furtado, A., Leitão, A., and Ramalho, G. (2004). Online Adaptation of Computer Games Agents: A Reinforcement Learning Approach. In *1st Brazilian Symposium on Computer Games and Digital Entertainment (SBGames'04)*.

Apperley, T. H. (2006). Genre and Game Studies: Toward a Critical Approach to Video Game Genres. *Simulation & Gaming*, 37(1):6–23.

Astrom, K. J. and Hagglund, T. (1995). *PID Controllers : Theory, Design, and Tuning*. Instrument Society of America.

Bernon, C., Camps, V., Gleizes, M.-P., and Picard, G. (2005). Engineering Adaptive Multiagent Systems: the ADELFE Methodology. In Henderson-Sellers, B. and Giorgini, P., editors, *Agent-Oriented Methodologies*, pages 172–202. Idea Group Pub.

Bethke, E. (2003). *Game Development and Production*. Wordware Publishing, Inc.

Bloom, B. S., Engelhart, M., Furst, E. J., Hill, W. H., and Krathwohl, D. R. (1956). Taxonomy of Educational Objectives: Handbook I: Cognitive Domain. *New York: David McKay*, 19(56).

Boes, J. (2014). *Apprentissage du contrôle de systèmes complexes par l'auto-organisation coopérative d'un système multi-agent*. Thèse de doctorat, Université de Toulouse, Toulouse, France.

Boubaker, S. and M'sahli, F. (2008). Solutions Based on Particle Swarm Optimization for Optimal Control Problems of Hybrid Autonomous Switched Systems. *International Journal of Intelligent Control and Systems*, 13(2):128–135.

Buche, C. and Querrec, R. (2011). An expert system manipulating knowledge to help human learners into virtual environment. *Expert Syst. Appl.*, 38(7):8446–8457.

Camps, V. (1998). *Vers une théorie de l'auto-organisation dans les systemes multi-agents basée sur la coopération : application à la recherche d'information dans un système d'information répartie*. Thèse de doctorat, Université Paul Sabatier, Toulouse, France.

Capera, D., Georgé, J.-P., Gleizes, M.-P., and Glize, P. (2003). The AMAS Theory for Complex Problem Solving Based on Self-organizing Cooperative Agents. In *12th IEEE Int. Workshops on Enabling Technologies, Infrastructure for Collaborative Enterprises*, pages 383–388.

Carpenter, G. A. (1989). Neural Network Models for Pattern Recognition and Associative Memory. *Neural networks*, 2(4):243–257.

Charles, D., Kerr, A., McNeill, M., McAlister, M., Black, M., Kcklich, J., Moore, A., and Stringer, K. (2005). Player-centred Game Design: Player Modelling and Adaptive Digital Games. In *Proceedings of the Digital Games Research Conference*, volume 285.

Colorni, A., Dorigo, M., Maniezzo, V., et al. (1991). Distributed Optimization by Ant Colonies. In *Proceedings of the First European Conference on Artificial Life (ECAL)*, volume 142, pages 134–142. Paris, France.

Csikszentmihalyi, M. (1990). *Flow: The Psychology of Optimal Experience*. NY: Harper and Row.

Cybenko, G. (1989). Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals and Systems*, 2(4):303–314.

Dayhoff, J. E. and DeLeo, J. M. (2001). Artificial Neural Networks. *Cancer*, 91(S8):1615–1635.

Dignum, V., Meyer, J.-J., and Weigand, H. (2002). Towards an Organizational Model for Agent Societies using Contracts. In *Proceedings of the First International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS): Part 2*, pages 694–695. ACM.

Dunn, R. S., Dunn, K. J., and Price, G. E. (1981). *Learning Style Inventory*. Price Systems New York, Lawrence, KS.

Dvorak, D. L. (1987). *Expert Systems for Monitoring and Control*. Artificial Intelligence Laboratory, The University of Texas at Austin.

Ferber, J. (1999). *Multi-agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Reading.

Fleming, P. J. and Purshouse, R. (2001). Genetic Algorithms In Control Systems Engineering. In *Proceedings of the 12th IFAC World Congress*, pages 383–390. Preprints.

Georgé, J.-P., Edmonds, B., and Glize, P. (2004). Making Self-Organising Adaptive Multiagent Systems Work. In Bergenti, F., Gleizes, M.-P., and Zambonelli, F., editors, *Methodologies and Software Engineering for Agent Systems*, chapter 6, pages 319–338. Kluwer, http://www.wkap.nl/.

Gleizes, M.-P., Camps, V., and Glize, P. (1999). A Theory of Emergent Computation based on Cooperative Self-organization for Adaptive Artificial Systems. In Ferrer Figueras, L., editor, *4th European Congress of Systems Science*. Spanish Society of System Science.

Glize, P. (2001). L'adaptation des systèmes à fonctionnalité émergente par auto-organisation coopérative. *Hdr, Université Paul Sabatier, Toulouse III*.

Golstein, J. (1999). Emergence as a Construct: History and Issues. *Emergence*, 1(1):49–72.

Goss, S., Aron, S., Deneubourg, J.-L., and Pasteels, J. (1989). Self-organized Shortcuts in the Argentine Ant. *Naturwissenschaften*, 76(12):579–581.

Guivarch, V. (2014). *Prise en compte de la dynamique du contexte pour les systèmes ambiants par systèmes multi-agents adaptatifs*. Thèse de doctorat, Université de Toulouse, Toulouse, France.

Hagan, M. T. and Demuth, H. B. (1999). Neural Networks for Control. In *Proceedings of the American Control Conference*, volume 3, pages 1642–1656 vol.3.

Hassan, R., Cohanim, B., De Weck, O., and Venter, G. (2005). A Comparison of Particle Swarm Optimization and the Genetic Algorithm. In *Proceedings of the 1st AIAA Multidisciplinary Design Optimization Specialist Conference*.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial intelligence*. University of Michigan Press.

Horn, P. (2001). Autonomic Computing: IBM's Perspective on the State of Information Technology. Technical report.

Jambu, S. (2013). Apprentissage de corrélations entre entrées et sorties de systèmes dynamiques. Rapport de master, Université de Toulouse, Toulouse, France.

Jennings, N. (2001). Building Complex Software Systems. *Communication of the ACM*, 44(4):35–41.

Jorquera, T. (2013). *An adaptive multi-agent system for self-organizing continuous optimization*. Thèse de doctorat, Université de Toulouse, Toulouse, France.

Kennedy, J. and Eberhart, R. (1995). Particle Swarm Optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948.

Kohonen, T. (1982). Self-organized Formation of Topologically Correct Feature Maps. *Biological Cybernetics*, 43(1):59–69.

Kolodner, J. L. (1992). An Introduction to Case-based Reasoning. *Artificial Intelligence Review*, 6(1):3–34.

Koster, R. (2005). *A Theory of Fun for Game Design*. Paraglyph Press.

Koza, J. R. (1992). *Genetic Programming: vol. 1, On the Programming of Computers by means of Natural Selection*, volume 1. MIT press.

Kubera, Y., Mathieu, P., and Picault, S. (2011). Ioda: an interaction-oriented approach for multi-agent based simulations. *Autonomous Agents and Multi-Agent Systems*, 23(3):303–343.

Lemouzy, S., Camps, V., and Glize, P. (2011). Principles and Properties of a MAS Learning Algorithm: a Comparison with Standard Learning Algorithms Applied to Implicit Feedback Assessment. In *International Conference on Intelligent Agent Technology (IAT)*, pages 228–235. CPS.

Li, B. and Riedl, M. O. (2011). Creating Customized Game Experiences by Leveraging Human Creative Effort: a Planning Approach. In *Agents for Games and Simulations II*, volume 6525 of *Lecture Notes in Computer Science*, pages 99–11. Springer-Verlag.

López-Ibáñez, M., Prasad, T., and Paechter, B. (2008). Ant Colony Optimization for Optimal Control of Pumps in Water Distribution Networks. *Journal of Water Resources Planning and Management*, 134(4):337–346.

Lotka, A. J. (1924). Elements of Physical Biology, Williams and Wilkins Co. *Inc., Baltimore*.

Magerko, B., Heeter, C., Fitzgerald, J., and Medler, B. (2008). Intelligent Adaptation of Digital Game-based Learning. In *Future Play*, pages 200–203.

Malone, T. (1981). What Makes Computer Games Fun? In *Proceedings of the Joint Conference on Easier and More Productive Use of Computer Systems. (Part - II): Human Interface and the User Interface*, CHI'81, page 143, New York, NY, USA. ACM.

Mathieu, P., Panzoli, D., and Picault, S. (2013). Virtual customers in a multiagent training application. In Pan, Z., Cheok, A., Müller, W., and Liarokapis, F., editors, *Transactions on Edutainment IX*, volume 7544 of *Lecture Notes in Computer Science*, pages 97–114. Springer Berlin Heidelberg.

Murphy, C. (2011). Why Games Work and the Science of Learning.

Niehaus, J. and Riedl, M. O. (2009). Scenario Adaptation: An Approach to Customizing Computer-Based Training Games and Simulations. In *Proceedings of the AIED 2009 Workshop on Intelligent Educational Games*, volume 3, pages 89–98.

Nussenbaum, E. (2004). Video game makers go hollywood. uh-oh.

Otterlo, M. and Wiering, M. (2012). Reinforcement Learning and Markov Decision Processes. In Wiering, M. and Otterlo, M., editors, *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pages 3–42. Springer Berlin Heidelberg.

Pfeifer, R. and Scheier, C. (2001). *Understanding Intelligence*. Bradford Books. MIT Press.

Picard, G. (2004). *Méthodologie de développement de systèmes multi-agents adaptatifs et conception de logiciels à fonctionnalité émergente*. Thèse de doctorat, Université Paul Sabatier, Toulouse, France.

Ram, A., Ontanon, S., and Mehta, M. (2007). Artificial Intelligence for Adaptive Computer Games. In *20th International Conference on Artificial Intelligence (FLAIRS'07)*. AAAI Press.

Ronald, E. M., Sipper, M., and Capcarrère, M. S. (1999). Testing for Emergence in Artificial Life. In *Advances in Artificial Life*, pages 13–20. Springer.

Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65(6):386.

Rougemaille, S. (2008). *Ingénierie des systèmes multi-agents adaptatifs dirigée par les modèles*. Thèse de doctorat, Université Paul Sabatier, Toulouse, France.

Russell, S. J., Norvig, P., Candy, J. F., Malik, J. M., and Edwards, D. D. (1996). *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Upper Saddle River, NJ, USA.

Schell, J. (2008). *The Art of Game Design – A Book of Lenses*. Elsevier/Morgan Kaufmann.

Sharma, M., Holmes, M., Santamaria, J., Irani, A., Isbell, C., and Ram, A. (2007). Transfer Learning in Real-time Strategy Games using Hybrid CBR/RL. In *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, pages 1041–1046, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Smith, B. (2002). Classical vs Intelligent Control. *Memorial University*.

Spronck, P., Ponsen, M., Sprinkhuizen-Kuyper, I., and Postma, E. (2006). Adaptive Game AI with Dynamic Scripting. *Machine Learning*, 63(3):217–248.

Stanley, K. O., Bryant, B. D., and Miikkulainen, R. (2005). Real-time Neuroevolution in the NERO Video Game. *Transactions on Evolutionary Computation*, 9(6):653–668.

Stephan, A. (1998). Varieties of Emergence in Artificial and Natural Systems. *Zeitschrift für Naturforschung. C. A journal of Biosciences*, 53(7-8):639–656.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.

Tsapanos, M., Chatzidimitriou, K. C., and Mitkas, P. A. (2011). A Zeroth-Level Classifier System for Real Time Strategy Games. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT'11)*, volume 2, pages 244–247.

Tyreus, B. D. and Luyben, W. L. (1992). Tuning PI Controllers for Integrator/Dead Time Processes. *Industrial & Engineering Chemistry Research*, 31(11):2625–2628.

van Ast, J., Babuska, R., and De Schutter, B. (2008). Ant Colony Optimization for Optimal Control. In *IEEE Congress on Evolutionary Computation (CCE'08)*, pages 2040–2046.

Venayagamoorthy, G. K. and Harley, R. G. (2007). Swarm Intelligence for Transmission System Control. In *IEEE Power Engineering Society General Meeting*, pages 1–4.

Videau, S., Bernon, C., Glize, P., and Uribelarrea, J.-L. (2011). Controlling Bioprocesses using Cooperative Self-organizing Agents. In *Advances on Practical Applications of Agents and Multiagent Systems (PAAMS)*, pages 141–150. Springer.

Volterra, V. (1927). *Variazioni e fluttuazioni del numero d'individui in specie animali conviventi*. C. Ferrari.

Vygotski, L. and Cole, M. (1978). *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press.

Weiss, G. (1999). *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press.

Westra, J., van Hasselt, H., Dignum, F., and Dignum, V. (2009). Adaptive Serious Games Using Agent Organizations. In Dignum, F., Bradshaw, J., Silverman, B., and van Doesburg, W., editors, *1st Int. Workshop on Agents for Games and Simulations (AGS'09)*, volume 5920 of *LNCS*, pages 206–220. Springer Verlag.

Westra, J., van Hasselt, H., Dignum, V., and Dignum, F. (2008). On-line Adapting Games using Agent Organizations. In Hingston, P. and Barone, L., editors, *IEEE Symposium on Computational Intelligence and Games (CIG'08), Perth, Australia*, pages 243–250. IEEE.

Willcoxson, L. and Prosser, M. (1996). Kolb's Learning Style Inventory (1985): Review and Further Study of Validity and Reliability. *British Journal of Educational Psychology*, 66(2):247–257.

Wilson, S. W. (1994). Zcs: A Zeroth Level Classifier System. *Evolutionary Computation*, 2(1):1–18.

Yannakakis, G. N. and Hallam, J. (2009). Real-Time Game Adaptation for Optimizing Player Satisfaction. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(2):121–133.

Zadeh, L. A. (1965). Fuzzy Sets. *Information and Control*, 8:338–353.

Zadeh, L. A. (2008). Is There a Need for Fuzzy Logic? *Information Sciences*, 178(13):2751–2779.

Ziegler, J. G. and Nichols, N. B. (1993). Optimum Settings for Automatic Controllers. *Journal of Dynamic Systems, Measurement, and Control*, 115(2B):220–222.

Zook, A., Lee-Urban, S., Riedl, M. O., Holden, H. K., Sottilare, R. A., and Brawner, K. W. (2012). Automated Scenario Generation: Toward Tailored and Optimized Military Training in Virtual Environments. In *Proceedings of the International Conference on the Foundations of Digital Games*, FDG'12, pages 164–171.

Zyda, M. (2005). From Visual Simulation to Virtual Reality to Games. *Computer*, 38(9):25 – 32.

List of Figures

List of Tables

Part III

Appendix

A Parametrized Functions to define criticality functions

A criticality function yields a value between 0 and 100 for any real value in input. It is defined in several parts as follows:

$$crit(x) = \begin{cases} 100 & \text{if } x < inf \\ \gamma_1 \frac{(x-\eta_1)^2}{2(\eta_1 - inf)} + \gamma_1(x-\eta_1) + \delta & \text{if } x < \eta_1 \\ -\gamma_1 \frac{(x-\eta)^2}{2(\epsilon_1 - \eta)} + \gamma_1(x-\eta) + \delta & \text{if } x < \epsilon_1 \\ 0 & \text{if } x < \epsilon_2 \\ -\gamma_2 \frac{(\eta_2 - x)^2}{2(\eta_2 - \epsilon_2)} + \gamma_2(\eta_2 - x) + \delta_2 & \text{if } x < \eta_2 \\ \gamma_2 \frac{(\eta_2 - x)^2}{2(sup - \eta_2)} + \gamma_2(\eta_2 - x) + \delta_2 & \text{if } x < sup \\ 100 & \text{if } x \geq sup \end{cases}$$

With $\gamma_1 = -2\frac{100}{\epsilon_1}$; $\gamma_2 = -2\frac{100}{100 - \epsilon_2}$; $\delta_1 = -\gamma_1 \frac{\epsilon_1 - \eta_1}{2}$; and $\delta_2 = -\gamma_2 \frac{\eta 2 - \epsilon_2}{2}$

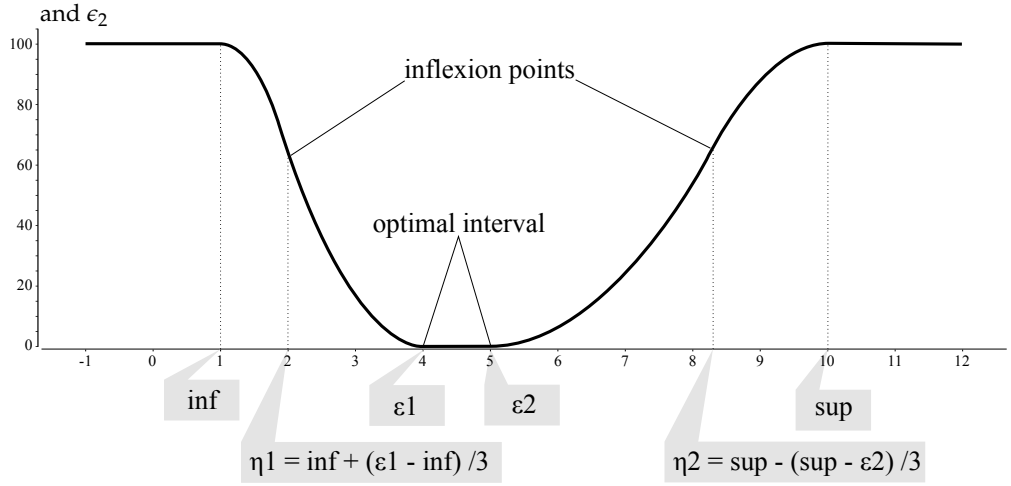
And $inf < \eta_1 < \epsilon_1 < \epsilon_2 < \eta_2 < sup$ being parameters that need to be given by experts:

- $[inf, sup]$ is the interval outside which the criticality is maximal,
- $[\epsilon_1, \epsilon_2]$ is the interval within which the criticality is null (i.e. the constraint or objective is completely satisfied),
- η_1 and η_2 are points in which the acceleration of the curve is changing, allowing the user to define more precisely the shape of the curve.

Figure A.1 shows an example of a criticality function with the values of the parameters that are used to define it. When defining a criticality function, users should at least define the $[inf, sup]$ interval, and the ϵ_1 value. Other values then take default values, for instance $epsilon_2$ is made equal to ϵ_1 reducing the optimal interval to a single value, and η_1 (resp. η_2) is defined as one third of the $[inf, \epsilon_1]$ interval (resp. $[sup, \epsilon_2]$) (see figure A.1). To rigorously define a criticality function, and still benefit from the intuitive expression described

at the beginning of this section, a human designer only has to specify a minimum of three parameters.

Figure A.1 — Example of criticality function defined with four parameters: inf, sup, ϵ_1 and ϵ_2



B Adaptive Value Tracker

The goal of an Adaptive Value Tracker (AVT) is to track a dynamic value in a given range, according to successive inputs of the type *greater* or *lower* [Lemouzy et al., 2011]. In addition to the tracked value, the AVT maintains a *delta*. This *delta* is used to modify the tracked value. When the input *lower* is received, the AVT subtracts *delta* from the tracked value. When the input *greater* is received, the AVT adds *delta* to the tracked value.

The strategy carried out by an AVT is adaptive because the *delta* value is variable. Each time two successive inputs are identical, the *delta* is increased to gain speed. Each time two successive inputs are opposite, *delta* is decreased to gain accuracy. Table B.1 summarizes the different possible *delta* modifications. It shows the calculation of *delta* at a time t (Δ_t) depending on the two previous inputs ($Input_t$ and $Input_{t-1}$), and on the previous delta value (Δ_{t-1}). Once the new delta is calculated, the new value of the AVT v_{t+1} is calculated by by adding ou substracting the delta valuer to the current AVT value depending on the current input ($Input_t$). Of course, the value cannot go outside the range, and the *delta* has both a minimal and a maximal value that depend on the size of the domain.

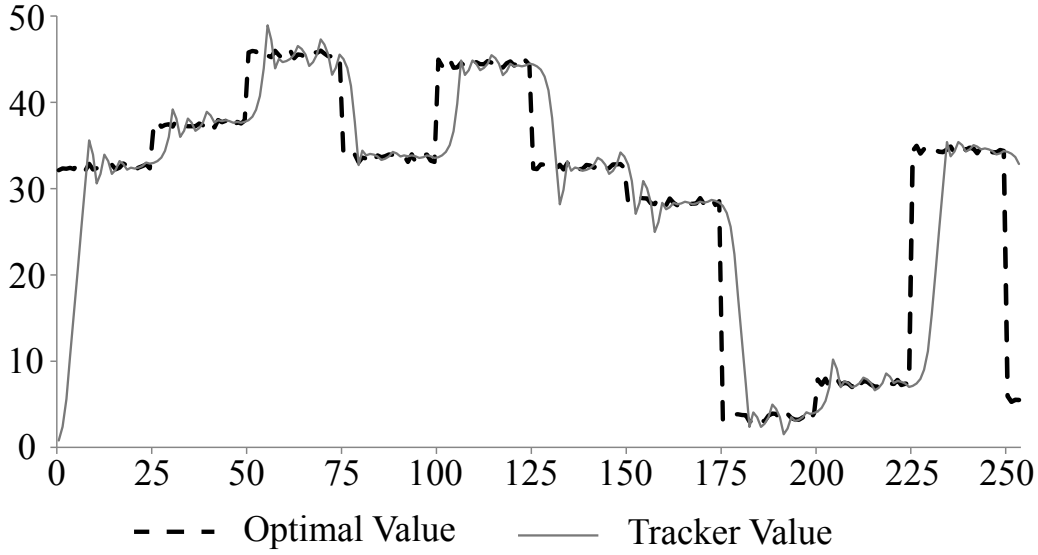
		$Input_t$	
		$\uparrow$	$\downarrow$
$Input_{t-1}$	$\uparrow$	$\Delta_t = \Delta_{t-1} \times 2$ $v_{t+1} = v_t + \Delta_t$	$\Delta_t = \Delta_{t-1}/3$ $v_{t+1} = v_t - \Delta_t$
	$\downarrow$	$\Delta_t = \Delta_{t-1}/3$ $v_{t+1} = v_t + \Delta_t$	$\Delta_t = \Delta_{t-1} \times 2$ $v_{t+1} = v_t - \Delta_t$

Table B.1 — Calculation of the AVT value, and delta at time t (v_t and Δ_t) based on the two previous inputs (either *greater* ($\uparrow$) or *lower* ($\downarrow$)), the previous value (v_t) and the previous delta (Δ_{t-1}).

This strategy allows to quickly move through the search space when the optimal value is far away from the current value, and at the same time, it allows for precise modification when the optimal value is close to the current value of the AVT. Figure B.1 shows an example of the behavior of an AVT. The solid line represents the evolution of the AVT value, and the dashed line shows the optimal value from which inputs are generated. For the generation of the optimal values, we generate a small random interval every 25 steps. For all of these 25 steps, the value is randomly taken in that interval. The result is a value that is greatly modified every 25 steps, but that is also never completely stable. We observe that despite

these variations, the AVT is able to quickly react, and to closely follow the optimal value.

Figure B.1 — Example of an AVT evolution



C Gambits Detailed Parameter List

During the Gambits project, work has been done with our industrial partner DCNS to identify all the aspects that could be used to modify a game experience so it reaches objectives defined by trainers. All the concepts that were to model in the game featured several parameters determining the way they should behave in the game.

All these aspects were listed in the form of tables of parameters. Each parameter is associated with a description, as well as with a range in which is value is valid.

Hereafter is a document used during the project to communicate with DNCS about these parameters. Their knowledge of the domain has been used to determine all relevant parameters, as well as realistic value ranges.

The document is written in French, given that it was the language of all partners in the project, but the non-French speaker can easily have a an idea of what the document contains. The document features explanatory texts and tables listing parameter names, descriptions, and ranges.

Paramètres et critères

Juillet 2011

Table des matières

1 Introduction

L'objectif de ce document est de lister les paramètres qui seront utilisés dans Gambits.

L'ensemble de ces paramètres définit la manière dont le jeu peut se dérouler. Ils concernent des propriétés relatives aux navires, mais également à l'environnement et aux ressources accessibles aux joueurs.

La liste est établie à partir des discussions ayant eu lieu au cours des précédentes réunions, ainsi qu'à partir des documents fournis par DCNS.

L'objectif est de lister les dépendances qui existent entre ces paramètres. Ensuite, un ensemble de critères seront définis pour permettre à un formateur d'exprimer des orientations souhaitées pour le jeu sans définir précisément les valeurs des paramètres.

Pour chacun des paramètres, si approprié, il s'agit d'associer une plage de valeurs.

La dernière étape est d'attribuer ces paramètres et critères aux rôles les plus pertinents, parmi:
- le game designer ;
- le level designer (aspect ludique) ;
- le formateur (aspect formation).

Les mécanismes d'ajustement des paramètres, couplés aux définitions de critères abstraits (voir §II), ont pour objectif d'offrir une assistance aux formateurs/designers.

Tels qu'ils sont définis, ces mécanismes permettront de procéder de plusieurs façons :
- La définition de scénario peut se faire avec un contrôle précis, en donnant une valeur à l'ensemble des variables présentes dans le scénario.
- A l'inverse, on pourra se contenter de définir seulement des objectifs abstraits, en laissant le soin au système de proposer les « valuations » les plus pertinentes, en fonction des performances du joueur, ou encore de sa motivation et de ses résultats.

Une approche à mi-chemin entre ces deux extrêmes semble appropriée. Un formateur/designer pourrait définir un scénario, voire même une session de jeu de manière abstraite, tout en contraignant, d'autre part, un sous-ensemble de paramètres.

Ajoutons que ces mécanismes sont relativement légers en termes de puissance de calcul et de mémoire nécessaires à leur gestion.

2 Paramètres de bas niveau

Pour chacun des paramètres, on liste :
- son nom ;
- son type, ainsi que son unité, si approprié ;
- sa plage de valeurs ;
- le rôle qui en a la responsabilité : **GD** (Game Designer) ou **LD/F** (Formateur/Level designer).

2.1 Paramètres relatifs aux navires

Pour chacune des 10 classes de navires :

Paramètre	Type	Plage de valeurs	Rôle
Nom de la classe	Cargo	N/A	GD
Vitesse maximum	20 (nœud)	0 à 20 nœuds (vitesse de croisière 18 nœuds)	GD
AIS (présence sur le navire?)	booléen	N/A	GD
RCS	10 (m^2)	2 à 10	GD

Paramètre	Type	Plage de valeurs	Rôle
Nom de la classe	Pétrolier	N/A	GD
Vitesse maximum	25 (nœud)	0 à 25 nœuds (vitesse de croisière 20 nœuds)	GD
AIS (présence sur le navire?)	booléen	N/A	GD
RCS	20 (m^2)	4 à 20	GD

Paramètre	Type	Plage de valeurs	Rôle
Nom de la classe	Croisière	N/A	GD
Vitesse maximum	25 (nœud)	0 à 25 nœuds (vitesse de croisière 25 nœuds)	GD
AIS (présence sur le navire?)	booléen	N/A	GD
RCS	20 (m^2)	4 à 20	GD

Paramètre	Type	Plage de valeurs	Rôle
Nom de la classe	Ferry	N/A	GD
Vitesse maximum	30 (nœud)	0 à 30 nœuds (vitesse de croisière 25 nœuds)	GD
AIS (présence sur le navire?)	booléen	N/A	GD
RCS	20 (m^2)	4 à 20	GD

Paramètre	Type	Plage de valeurs	Rôle
Nom de la classe	Chalutier	N/A	GD
Vitesse maximum	15 (nœud)	0 à 15 nœuds (entre 2 et 3 nœuds en pêche et 15 nœuds en marche normale)	GD
AIS (présence sur le navire?)	Booléen (oui si longueur supérieure à 12 mètres)	N/A	GD
RCS	4 (m^2)	1 à 4	GD

Paramètre	Type	Plage de valeurs	Rôle
Nom de la classe	Thonier	N/A	GD
Vitesse maximum	15 (nœud)	0 à 15 nœuds (en pêche quand la senne est déployée la vitesse est 0 pendant environ 2 heures). Marche normale 15 noeuds TBD	GD
AIS (présence sur le navire?)	Booléen (oui si longueur supérieur à 12 mètres).	N/A	GD

RCS	4 (m^2)	1 à 4	GD

Paramètre	Type	Plage de valeurs	Rôle
Nom de la classe	Bâtiment de guerre	N/A	GD
Vitesse maximum	35 (nœud)	0 à 35 nœuds TBD	GD
AIS (présence sur le navire?)	Booléen (oui mais souvent éteint)	N/A	GD
RCS	15 (m^2)	3 à 15	GD

Paramètre	Type	Plage de valeurs	Rôle
Nom de la classe	Embarcation rapide	N/A	GD
Vitesse maximum	45 (nœud)	0 à 45 nœuds (vitesse de croisière 40 nœuds)	GD
AIS (présence sur le navire?)	Booléen (non pas d'AIS à bord)	N/A	GD
RCS	1(m^2)	0,5 à 1	GD

Paramètre	Type	Plage de valeurs	Rôle
Nom de la classe	Voilier	N/A	GD
Vitesse maximum	18 (nœud)	0 à 18 nœuds (le profil de vitesse est assez chahuté car dépend des risées de vent)	GD
AIS (présence sur le navire?)	Booléen (non pas d'AIS)	N/A	GD
RCS	1,5 (m^2)	0,5 à 1,5	GD

Paramètre	Type	Plage de valeurs	Rôle
Nom de la classe	Yatch	N/A	GD
Vitesse maximum	35 (nœud)	0 à 35 nœuds (vitesse de croisière 25 nœuds)	GD
AIS (présence sur le navire?)	Booléen (oui)	N/A	GD
RCS	5 (m^2)	1 à 5	GD

Pour chaque navire individuellement :

Paramètre	Type	Plage de valeurs	Rôle
Nom du navire	Chaîne de caractères	N/A	GD + F/LD (génération aléatoire possible).
Pavillon	Chaîne de caractères	N/A	GD + F/LD (génération aléatoire possible)
MSSI	Chaîne de caractères composée d'une suite de neuf chiffres (il est unique pour chaque navire)	N/A	GD + F/LD (génération aléatoire possible)
Navire black-listé ?	Booléen	N/A	F/LD

Attitude coopérative ? (c'est-à-dire que le navire possède un émetteur AIS ou VMS (navire de pêche)	Booléen	N/A	F/LD

2.2 Paramètres relatifs aux ressources accessibles au joueur

	Paramètre	Type	Plage de valeurs	Rôle
Radars	Nombre	Entier	TBD	GD + F/LD
	Portée	Entier (miles)	20 miles	GD + F/LD
	Probabilité de panne	Réel	[0 ; 1]	GD + F/LD
Avions	Nombre	Entier	2 à 3	GD + F/LD
	Autonomie	Entier (miles)	4 à 5 heures	GD + F/LD
	Portée radar	Entier (miles)	100 miles	GD + F/LD
	Portée visuelle	Entier (miles)	15 miles	GD + F/LD
Patrouilles	Nombre	Entier	TBD	GD + F/LD
	Autonomie	Entier (miles)	2 à 3 jours	GD + F/LD
	Portée radar	Entier (miles)	20 miles	GD + F/LD
	Portée visuelle	Entier (miles)	5 miles	GD + F/LD
VMS	Portée VMS	Entier (miles)	N'importe où dans le monde car les données sont transmisses du navire à un centre à terre par des satellites de télécommunication	
AIS	Portée AIS	Entier (miles)	Dépend de l'altitude à laquelle est placé le récepteur AIS à la côte (on peut faire un profil entre portée de 20 miles si récepteur à l'altitude 0 et 100 miles si le récepteur à 500 m d'altitude)	GD + F/LD

2.3 Paramètres relatifs à l'environnement

Paramètre	Type	Plage de valeurs	Rôle
Force du vent	Entier (force)	1 à 12 (voir ci-dessous la définition de l'échelle Beaufort)	F/LD
Hauteur des vagues	Entier (mètre)	0.1 à 10 mètres	F/LD
Visibilité	Entier (miles)	0 à 5 miles (voir ci dessous l'échelle de	F/LD

		visibilité utilisée en météo marine)	

Définition de l'échelle Beaufort :

Force	Appellation	noeuds	km/h	Effets observés sur mer	Effets observés sur terre
0	Calme	< 1	< 2	La surface de la mer est unie comme un miroir, mais pas forcément plane.	La fumée s'élève verticalement.
1	Très légères brises	1 à 3	2 à 6	Il se forme des rides ressemblant à des écailles de poisson, mais sans écume.	La fumée, mais non la girouette, indique la direction du vent.
2	Légères brises	4 à 6	7 à 11	Vaguelettes courtes mais plus accusées. Leur crête a une apparence vitreuse mais elles ne déferlent pas. Par bonne visibilité, la ligne d'horizon est toujours très nette.	A 4 noeuds, on commence à sentir le vent sur le visage; les feuilles frémissent et les girouettes bougent.
3	Petite brise	7 à 10	13 à 19	Très petites vagues. Les crêtes comment à déferler. Écume d'aspect vitreux. Parfois quelques moutons épars.	Feuilles et brindilles bougent sans arrêt. Les petits drapeaux se déploient.
4	Jolie brise	11 à 16	20 à 30	Petites vagues devenant plus longues. Présence évidente de moutons.	Poussières et bout de papier s'envolent. Les petites branches sont agitées.
Avertissement aux petites embarcations (> 20 noeuds)					
5	Bonne brise	17 à 21	31 à 39	Vagues modérées prenant une forme plus nettement allongée. Formation importante de moutons, premier détail visible sur le plan d'eau. Parfois quelques embruns.	Les petits arbres feuillus se balancent. De petites vagues avec crête se forment sur les eaux intérieures.
6	Vent frais	22 à 27	40 à 50	De grosses vagues, ou lames, commencent à se former. Les crêtes d'écume blanche sont parfois plus étendues. Les embruns sont fréquents.	Les grosses branches sont agitées. On entend le vent siffler dans les fils téléphoniques et l'usage du parapluie devient difficile.
7	Grand frais	28 à 33	51 à 61	La mer grossit. L'écume blanche qui provient des lames déferlantes commencent à être soufflée en traînées qui s'orientent dans le lit du vent.	Des arbres tout entiers s'agitent. La marche contre le vent devient difficile.

Avertissement de coup de vent (> 34 noeuds)					
8	Coup de vent	34 à 40	62 à 74	Lames de hauteur moyenne et plus allongées. De la crête commencent à se détacher des tourbillons d'embruns. Nettes traînées d'écume orientées dans le lit du vent.	De petites branches se cassent. La marche contre le vent devient presque impossible.
9	Fort coup de vent	41 à 47	75 à 87	Grosses lames. Epaisses traînées d'écume dans le lit du vent. La crête des lames commence à vaciller, s'écrouler et déferler en rouleaux. Les embruns peuvent réduire la visibilité.	Peut endommager légèrement les bâtiment (bardeaux de toitures).
Avertissement de tempête (> 48 noeuds)					
10	Tempête	48 à 55	88 à 102	Très grosses lames à longues crêtes en panache. Épaisses traînées d'écume. La surface des eaux semble blanche. Le déferlement en rouleaux devient intense et brutal. Visibilité réduite.	Déracine les arbres et endommage sérieusement les bâtiments.
11	Violente tempête	56 à 63	103 à 117	Lames exceptionnellement hautes. Mer complètement recouverte de bancs d'écume. Visibilité réduite.	Dégâts considérables.
12	Vent d'ouragan	64 à 71	118 à 132	L'air est plein d'écume et d'embruns. La mer est entièrement blanche, du fait des bancs d'écume dérivants. Visibilité très fortement réduite.	Rare

Définition de l'échelle de visibilité

Qualification de la visibilité	Distance en miles
Bonne	Supérieure à 5 miles
Médiocre	Entre 2 et 5 miles marins
Mauvaise	Entre 0,5 et 2 miles
Brouillard	Inférieure à 0,5 miles

3 Critères abstraits

On définit des critères qui sont des abstractions proposées au formateur afin d'exprimer de manière plus aisée des caractéristiques de tours et sessions de jeu sans avoir à donner une valeur précise à l'ensemble des paramètres de bas niveau. Ces critères sont :

- La qualité des ressources : l'efficacité des moyens d'observations mis à la disposition du joueur. Pour l'ensemble des capteurs disponibles, ce critère concernera leur portée, leur puissance, ainsi que leur résistance aux pannes.
- La quantité de ressources : mesure quantitative des moyens d'observation. Le nombre des équipements, mais également leur variété, afin de proposer une tenue de situation plus efficace, et plus résistante aux éléments perturbateurs (venant de l'environnement, ou du comportement des navires).
- La difficulté liée à l'environnement : l'ensemble des caractéristiques définissant l'environnement peut avoir une influence sur la difficulté du jeu. Principalement, en limitant l'utilisation ou l'efficacité des moyens d'observations. La force du vent peut exclure l'utilisation des avions, la visibilité limite les informations recueillies par les patrouilles et la hauteur des vagues réduit l'efficacité des radars.
- L'efficacité des contrevenants : en plus du comportement adopté par les contrevenants, certaines de leurs caractéristiques propres peuvent influer sur la difficulté qu'aura le joueur à les détecter. Un navire plus

efficace pourra, par exemple, avoir une RCS plus faible, ne sera pas black-listé, pourra avoir une plus grande vitesse maximale, ou encore ne pas adopter d'attitude coopérative.

Certains critères peuvent servir à exprimer la nature de la population de navires, ainsi que leurs répartitions dans les sessions de jeu, sans avoir à définir manuellement l'ensemble des dates de création :
- Population de neutres.
- Population de navires pour une infraction donnée.
- Population totale.
- Proportion de contrevenants.

De plus, des critères pourront servir à exprimer l'organisation des sessions de jeu, toujours sans avoir besoin de définir explicitement l'ensemble des paramètres nécessaires à la préservation de la cohérence des sessions de jeu :
- Temps total de la session.
- Nombre de tours.
- Nombre de vagues dans chacun des tours.

Enfin, les critères peuvent servir à exprimer des performances mesurables correspondant aux objectifs que l'on fixe au joueur.
Typiquement, on imagine des mesures permettant d'estimer les performances du joueur telles que :
- Taux global d'infractions repérées.
- Taux spécifique d'infractions repérées (pour un type d'infractions particulier).
- Taux de faux positifs.
- Temps de réaction.
- Stabilité dans le temps.
- Endurance (capacité de maintenir un niveau de compétence sur la longueur).

Ces critères ne sont pas nécessairement chiffrés. Un formateur peut vouloir simplement exprimer les points qu'il souhaite faire travailler au joueur, et laisser au système le soin de valuer l'ensemble des critères de performances.

4 Conclusion

En conclusion, rappelons que ces mécanismes offrent une grande souplesse.

La possibilité de tout contrôler est laissée au formateur, tout en lui donnant la possibilité de n'exprimer que des exigences de très haut niveau.

De plus, tous les niveaux entre ces deux extrêmes sont utilisables. On pourra fixer fortement certains aspects du jeu, en laissant au système le soin de régler le reste.

À la charge de l'utilisateur de manipuler le système comme il l'entend.