Universidad Rey Juan Carlos
Departamento de Arquitectura y Tecnología de Computadoras,
Ciencias de la Computación e Inteligencia Artificial

Gaming with Emotions: An Architecture for the Development of Mood-Driven Characters in Video Games

Thesis defended by

Luis Peña

for the degree of
PhD in Computer Science (Doctor en Informática)

Supervised by
Prof. Dr. Sascha Ossowski & Prof. Dr. Jose M Peña

Madrid, España
Junio 2013

Don **Sascha Ossowski**, Catedrático del Departamento de Arquitectura y Tecnología de los Computadores, Ciencia de la Computación e Inteligencia Artificial de la Universidad Rey Juan Carlos.

AUTORIZA:

La presentación de la Tesis Doctoral titulada:

Gaming with Emotions: An Architecture for the Development of Mood-Driven Characters in Video Games

Realizada por **Don Luis Peña Sánchez** bajo su inmediata dirección y supervisión en el Departamento de Arquitectura y Tecnología de los Computadores, Ciencia de la Computación e Inteligencia Artificial y que presenta para la obtención del grado de Doctor por la Universidad Rey Juan Carlos.

Móstoles, 19 de junio de 2013

Don **José María Peña**, Profesor Titular de Universidad del Departamento de Arquitectura y Tecnología de Sistemas Informáticos de la Universidad Politécnica de Madrid.

AUTORIZA:

La presentación de la Tesis Doctoral titulada:

Gaming with Emotions: An Architecture for the Development of Mood-Driven Characters in Video Games

Realizada por **Don Luis Peña Sánchez** bajo su inmediata dirección y que presenta para la obtención del grado de Doctor por la Universidad Rey Juan Carlos.

Boadilla del Monte, 20 de junio de 2013

Thesis advisors
Prof. Dr. Sascha Ossowski & Prof. Dr. Jose M Peña

Author
Luis Peña

Gaming with Emotions: An Architecture for the Development of Mood-Driven Characters in Video Games

Abstract

In the present dissertation, we study the emotional component of the behavior of artificial characters in video games. Our primary aim is to improve the video game playing experience by increasing the sense of realism in gaming scenarios.

For this purpose, we develop an emotion simulation model called EEP that accounts for the impact of external events on a character's mood state, and analyze its relevance for the development of *mood-driven* behaviors as part of the control strategies of artificial characters. In addition, we provide a mechanism that improves the development procedure of video game characters, by developing a new hybrid machine learning model called WEREWoLF that purposefully combines reinforcement learning and evolutionary techniques, so as to automatically generate character control strategies associated to different mood states. Both models are integrated into the AGCBAR architecture, which constitutes the solution proposed in this dissertation to the problem of efficiently designing mood-driven strategies for artificial characters in video games. The AGCBAR architecture is capable of encompassing a broad variety of game engine cores, and is thus applicable to a wide spectrum of video games.

We assess the adequacy of the above architecture and its components in different ways. While the EEP model has been evaluated on the basis of the judgment of expert gamers, the WEREWoLF algorithm has undergone a quantitative evaluation in a video game scenario. Finally, we implement the complete architecture together with an experimental video game framework in a complex case study, comparing the development effort of mood-driven artificial characters using the AGCBAR architecture together with EEP and WEREWoLF, to traditional implementation techniques.

Acknowledgments

Completing this doctoral work has been a wonderful and often overwhelming experience. It would not have been possible to write this doctoral thesis without the help and support of the kind people around me, to only some of whom it is possible to give particular mention here. But, I want to extend all my thanks to all of them.

First, this thesis would not have been possible without the help, support and patience of my supervisors, Prof. Dr. Sascha Ossowski and Prof. Dr. José-María Peña. Their advice have been invaluable on both academic and personal level, for which I extremely grateful.

I would like to acknowledge the support of the Artificial Intelligence Group of the Universidad Rey Juan Carlos, they make possible my research and the success of this work was mostly done thank to their financial, technical and human support.

Thanks also to all people from the Game Intelligence Group at the University of Essex for hosting me there. I really appreciate their help, their warm welcome, and their support.

Loads of thanks to the people that review and comment my work. Thanks Diego Fradejas for his development in the vBattle interface and his valuable ideas about the game. And, Carlos Millan for his aid in the development of the storytelling scenario.

Now, I want to include some words in Spanish:

Gracias a todos mis amigos que he dejado de ver por realizar esta tesis y que, aún así, me siguen hablando.

Quiero agradecer muy especialmente a toda mi familia por su incondicional apoyo, ellos han hecho posible mis largas horas de trabajo, animándome y ayudándome contínuamente. ¡Gracias a todos!

Papá, mamá, ¡muchas gracias por estar ahí en todo momento! Gracias hermano por animarme de principio a fin en esta nueva etapa, por tus consejos y por tus ideas.

Y, finalmente, ¡gracias Vero! tu has estado a mi lado desde que pensé empezar este trabajo y siempre me has apoyado, en mis éxitos y en mis fracasos. A ti te debo la fuente más importante de energía para este trabajo, nuestros fantásticos hijos: Claudia y Darío.

to Vero,
Claudia,
and Dario.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> It should be noted that the games of children are not games, and must be considered as their most serious actions
>
> Michel de Montaigne

1.1 Motivation

1.1.1 Why making a video-game related thesis?

From a business perspective, the foremost reason for doing research in the field of video games is their ever-growing economic relevance.[1]

The last market trend report "Gaming Ecosystem 2011" [2] states that hardware and game sales exceeded US$74.000 millions in 2011, which exceeds the US$67.000 millions from 2010. The video game industry is the major entertainment business since the last decade, outperforming then cinema and the music ones. As clearly shown by the CBS analysis: only the video game contents (considering them as the final game product) represent a US$23.000 millions of economic profit that is three times bigger than the music industry gains (US$6.900 million) and double than the cinema (US$10.600 million). In the context of software development, the numbers also show that video game development represents not only the most profitable branch in the ICT-related industry but also the largest in terms of gross income.

Nevertheless, as a software development industry, video game development has its own particular shortcomings: it is constrained by very short software development cycles in which the activities of graphics and music designers, story script writers, or final editors,

[1] http://www.theesa.com/facts/pdfs/VideoGames21stCentury_2010.pdf

[2] http://www.gartner.com/newsroom/id/1737414

represent a significant amount of time and resources, compared to the time and resources spent by the developers and programmers [8]. In such scenario, the software development is restricted by the appropriate adjustment of some particular behaviors, like the design of artificial intelligence controllers, the programming of graphical effects and the typing of all the different pieces of the project together. This kind of development process deeply relies on the existence of powerful and complete frameworks to support rapid and effective development [70]. These frameworks include graphical engines, event control mechanisms, templates for control mechanisms, basic search, and planning and decision algorithms to name only a few of the most relevant aspects.

Nowadays, the video game industry has evolved towards more realistic games, not only in terms of the visual and graphic qualities, but also in terms of the behavior of the different characters of the game. In the last few years, Massive Multiplayer On-Line Games (MMOGs), such as *World of Warcraft*, emerged to satisfy the needs of more exigent Gamers (game players) that demanded that the new characters in their games were something more than carbon-copy versions of stereotypes. To respond to such consumer demands, playing with other human players offers the thrill of interesting enemies or allies with surprising reactions to the events of the game, a kind of behavior not fully achieved by virtual players controlled by artificial intelligence game controllers. These MMOGs have represented a revolution that has improved the gaming experience and the satisfaction of a large number of players interacting in the virtual world.

Although the MMOGs have boosted the industry in the last few years[3], the Newzoo's trend report shows that their market is growing in terms of number of games, amount of people paying and the money spent. But, the gamers also demand more advanced games that require aspects far beyond more realistic graphics. For them, there are considerations of great importance in aspects such as human-like behavior, adaptive strategies of the opponents, on-the-fly level and scenario generation or open interaction with the agents and the environment.

On top of that, the video game industry, and in particular the European industry has to compete with huge companies in the US or Asia[4], companies that are able to recruit large teams of designers and developers for the creation of the next versions of any of the AAA class games (big budget video game productions) that each year are issued worldwide. These teams can develop not only the contents for the new games but also the additional resources to update or even re-design the software platform they use.

In today's video game development field, the success factor has not only been the mere technological supremacy but mostly the intuition and the creativity of the game designers in the conception or in the approach of the game. Here is an example that clearly

[3]http://www.newzoo.com/trend-reports/mmo-trend-report/

[4]http://www.b105.fi/egdf/wp-content/uploads/2011/06/EGDF-Policy-papers-2nd-edition-Game-Development-and-Digital-Growth-web.pdf

illustrates this situation[5]: from the three major home video consoles, Sony PS3 has been the most advanced in terms of the core technology underneath, followed by Microsoft Xbox and far away from Nintendo Wii. Even though the technological possibilities of the Sony platform outperform the Nintendo alternative, Wii developers have succeeded in the design of highly participative and interactive video games that have opened new markets and rocketed Nintendo's market share to 27% more than Sony's share in its segment. This has been supported by the design of new interactive game devices that have made the brilliant ideas of new game concepts possible.

Thus, if we observe the budget and the sale figures of this industry, it is easy to infer the importance and relevance in terms of investments and, so then, the interest in research in the different fields of the computer science that can be applied successfully in their projects[6].

Why writing an AI video game thesis?

As reported by Ralph Edwards' article on the economics of games[7], the necessity of high-quality graphics, complex content and richer storytelling and scripting have made:

> *... the size of teams required to make games for the newer consoles doubling when compared to the previous generation, particularly with the number of modelers, animators, and other artists now needed, you can see why the cost of development keeps making significant jumps for each subsequent new generation of consoles.*

Thus, today's AAA class games require sophisticated character models. Still, while in such games the visual aspects of the virtual characters are usually well polished, their behavior has only recently been considered as important as the visual effects. In the past few years, game development has applied static off-the-shelf solutions that sometimes led to inflexible and unrealistic behaviors. Of course, this could cause players to become dissatisfied with the game: they wanted to have the feeling that the characters in the scenario had a purpose and a goal, and were not just roaming around the scene. Today, different planning and reasoning techniques have been adapted to produce more effective (rational) behaviors of the virtual characters as well as more complex interactions with them. Still, in order for the virtual characters to be more realistic and believable, they should also be (sometimes) surprising and challenging for the player, not always making fully rational decisions, but also the ones that are motivated by an emotional response to the environment and the recent events. This task grows in complexity as both the number of different game agents and the number of possible actions increase.

[5] http://www.vgchartz.com/tools/hw_date.php?reg=Global&ending=Yearly

[6] http://www.polygon.com/2012/10/1/3439738/the-state-of-games-state-of-aaa

[7] http://www.ign.com/articles/2006/05/06/the-economics-of-game-publishing

Even when the graphic quality of a video game is one of the major issues for selling a new product, the fidelity of the users, buying new editions year after year, relies on other aspects such as the story script or the playing experience. A key factor in this playing experience is the behavior of the agents in the game[33]. We must consider that the behavior of the agents, as well as the rest of the aspects of the game, are developed inside of a short-term highly-demanding industry. Putting all these considerations together, the design of appropriate agent controllers in the least-expensive manner is a requirement of modern video game design.

As it was previously stated, besides the graphics, the success of a game also depends heavily on other aspects such as the story script or the playing experience. A key factor of the playing experience is the behavior of the agents in the game, which are controlled by the artificial intelligence routines (agent controllers and planners)[91]. State-of-the-art artificial intelligence in video games has greatly evolved in the last few years. More advanced intelligence techniques have substituted the traditional rule-based controllers. Although these new methods have improved the quality of new controllers, they require a lot of hand coding and tuning efforts by the developers. Moreover, the development of more believable and human-like behaviors are still pending issues in today's video games.

The industry needs to adopt a new technology in order to achieve these results in a constrained time frame, derived from the timings and work flows of their development[17]. Unfortunately, the industry, pushed by market deadlines, invests less than it would be desirable on the research of new advances in this field. The reason behind this is that it is important for the industry to speed-up the development time of the characters' controllers reaching the best quality of these controllers in a limited amount of time. They tend to apply those well-established procedures to develop existing controllers.

Nevertheless, even when the industry follows the aforementioned path, there is a huge opportunity to conduct active research on adaptive and tailored artificial intelligence mechanisms, focused on two major objectives **(1) to create more believable character behavior and (2) to assist game design in the coding of intelligent game controllers**.

1.2 Objectives

This thesis is motivated by the insight that the success of a new video game product does not only rely on pure technology but also on the ideas that make game experience challenging, addictive and new. Therefore, we set out from the following hypothesis:

> *It is possible to facilitate the development of video game controllers for the virtual characters applying learning techniques and personality traits.*

Furthermore, the automatic or computer-assisted creation of controllers and the application of emotional models to characters' behaviors can provide an improvement in

the design of the simulated environments provided by the video games. Thereby, it can introduce a new factor to bring sense of a more realistic environment and a better player experience through the support tools that can be included in the development cycle of the video games, producing more believable characters. Thus, this thesis pursues a set of objectives:

1. After considering the different open issues that can provide an improvement in the development phase of the entertainment software, there are many **focused on the character's behavior** (aside from their visual appearance and animation). The *main goal* of this thesis is to come up with methods and mechanisms that improve this specific software development on:

   (a) **The automatic creation of controllers that produce strategies that can be used, at least as baseline, for the video game characters**. The programming of character controllers is time consuming and error-prone, so mechanisms that can automatically provide some baseline behaviors in certain scenarios could save many resources in the development phase.

   (b) The sense of realism that can lead to a richer experience has primordial importance on the development of new video games. Photo realistic graphics (which have greatly improved) must be complemented with believable behaviors. Thus, we identify **the necessity of dealing with the emotional characteristics and personality profiles** of the characters influenced by the perception of the environment.

   Therefore, to achieve this objective, we propose the **creation of a model or architecture that can introduce a formalized and standardized mechanism** with these two features.

2. The architecture must be flexible enough to allow the application of a variety of components that can bring up different implementations of controllers and emotional models, so it can be used by different approaches whenever they are compliant with the requirements of the architecture. It makes **the model more applicable and extensible** getting profit from the knowledge of the different development teams, which are usually specialized in some techniques that can be applied to the development through this architecture.

3. In terms of experimentation and applicability, another main goal is to **create a set of techniques, models and algorithms** that can be applied on the proposed architecture **which meet the requirements of the field of application**, not only on the computational aspects, but on the final objective of making more believable characters that can be controlled by automatically created behaviors. Therefore, we present the objective of creating an algorithm that can build strategies in complex environments

(as any scenario of a commercial video game) guided by different goals; these goals are set by the final objectives endowed by the current mood of the character. Thus, the character must "experience" the emotions produced by the environment events so he can reach different moods. We also need to create an emotional model that is flexible enough to be generally used in the video game environments, and that is powerful enough to represent faithfully the emotional mechanisms.

4. Finally, we find of crucial importance the **demonstration of the architecture and the models in the real environment of video game development**, focusing our attention on the constraints of the development process of this kind of software which are mainly time and complexity. The architecture and the models that we propose must be tested on frameworks or products that validate their application in future developments.

1.3 Methodology

This thesis addresses the objectives described above: *creating an architecture* that can include the components necessary to support the *automatic creation of controllers* for video game characters, adding the *emotional behavior* to increase the believability in these characters.

Therefore, we apply a top-down approach in the design of the solution.

① We formulate a general description of the components that we want to include (detailed in Section 1.3.1).

② The different components are analyzed in a bottom-up approach, because we want an architecture applicable to the specific field of video game design and production, so we need to extract the key features of the application environment. This phase details the processes expected to be performed by each of the components according to a set of pre and post conditions: *an abstract specification of the procedural components* (these steps are described in Sections 1.3.2 and 1.3.3).

③ We establish the necessary information exchange interfaces that grant the intercommunication between the components across the architecture. At this point, we describe the input data required by the components and the output data produced by the components.

With the aforementioned three steps, we define the architecture proposed as solution (fully described in Section 1.3.4).

④ As an example, we implement the architecture in a video game framework to apply the techniques to create the automatic controllers and to provide the emotional behavior in a game scenario.

⑤ We validate each of the implemented components of the architecture following the specifications, and we also test each of them independently.

⑥ Finally, we test the integration of the components to provide a complete solution in the context of a video game.

1.3.1 Architecture Design

For the architecture design, we decompose the whole solution in three major components: the game engine, the mechanism to simulate the emotions and the support for the automatic creation of controllers.

The first element, the game engine, represents the basic component of any video game. It encloses the logic, rules and features of the game itself. This element maintains the environment where the characters are at. For the experimental part of the thesis, we propose to use both an existing commercial game engine, wrapped to match the expected interfaces, and a custom game engine with configurable levels of details as well.

The simulation of emotional responses is substantiated by a component that contains the necessary structures to ensure its credibility. These structures are the emotional models, which have their basis in applied psychology. We analyze some models from cognitive psychology to extract its characteristics, so we can apply them in our architecture.

The creation of the controllers can be addressed by different approaches. The soft computing approach arises as a feasible mechanism. The automatic exploration of the possible solutions to a proposed problem is pretty similar to the way that a character can learn to play a game. These techniques explore different strategies in different scenarios trying to maximize their expected outcome as result of the application of the rules of the game. We study the novel combination of soft computing and reinforcement learning algorithms as a possible solution as each of them has previously been applied on a wide set of problems. From these algorithms, we select the features that the architecture must provide in order to enable the learning of strategies by the characters involved in a video game.

Therefore, with the features extracted from the different models we specify the interfaces that are defined to ensure the communication among the different components. These interfaces, characteristics and constraints, are specified for the component implementations.

1.3.2 Emotional Model Implementation

With the specifications extracted during the design of the architecture we implement an emotional model that can act as the component in charge of the emotional responses. The characters using the architecture must take the changes produced in the environment as feedback to produce their emotional responses according to the simulation model implemented by the component.

As an exemplification of the usage of the proposed emotional model, we test it on a small scenario where the emotional responses can be seen in action. The emotional model is flexible enough to make possible the development of this scenario on top of the aforementioned commercial video game engine.

Therefore, the emotional model is presented as a deliverable component that matches the parameters, structures and interfaces to be included in the proposed architecture.

1.3.3 Learning Algorithm Implementation

As in the case of the emotional model, we implement an algorithm that meets the requirements of the architecture and environment. The algorithm must be applicable to the video game environment. Thus, the learning phase length, the complexity of the environment and the diversity of actions are problems for the application of many of the standard learning algorithms. Furthermore, a challenging research question it is the application of an approach based on a combination of different techniques of computation learning.

Henceforth, a new hybrid algorithm is tested as a solution to this kind of problems. We create a customized version of a video game and prove the convergence of the learned strategies to a competitive solution generated in an automatic way for the case of this game engine.

Finally, the interfaces and features required by the automatic controller procedure referred by the architecture are fulfilled by the implementation of this new learning algorithm.

1.3.4 Integration of the Architecture Components

Finally, we include the two components in the architecture. We extend the custom video game framework to test the integration of the elements with a game engine. The game should be complex enough to represent the characteristics of, at least, one genre of video game. Moreover, the characters and environments included in this video game should be rich enough to display the learned strategies and the emotional responses addressed by this thesis.

As a matter of fact, this part of the methodology tackles the cost-effectiveness of the approach presented by this thesis. Once each of the components have been independently validated, the integration of both has to be tested. In this sense, the comparison of the resources required to fulfill the development of the character controllers is performed between a standard hand-coded version against the assisted way for a fully-integrated strategic and emotional control set.

1.4 Structure of the Thesis

The thesis is structured as follows:

1. Chapter 2 provides a survey of the state of the art in the different research fields that this thesis is draws upon; in particular:
   - emotional models developed in different scenarios for virtual characters. We analyze the psychology rationale underlying these models and the main parts needed for a well founded model for the application of emotions.
   - soft computing approaches that can be used for automatic controller creation and that we use as basis for the design of the Learning Engine implemented in the proposed architecture.
2. In Chapter 3, we present the AGCBAR Architecture developed in order to achieve the main objectives pursued by this thesis. We illustrate the different restrictions and interfaces needed by components that implement the different parts of the architecture. Later on, we present a set of guidelines for easing the application of the architecture in different scenarios.
3. Chapter 4 describes the definition, implementation and validation of the EEP Model which is used as a sample implementation of an Emotional Engine that can be used in the AGCBAR Architecture. In order to carry out the validation, we have considered as set of independent tests to evaluate EEP model, outside of AGCBAR Architecture.
4. Next, Chapter 5 presents a proposed Learning Engine (called WereWoLF) implemented for the architecture. As in the case of the EEP Model, the preliminary validation is made outside from the AGCBAR Architecture.
5. The experimental results of the AGCBAR Architecture, presented at Chapter 6, were obtained by the implementation of the different components in a video game framework named vBattle. The implemented scenario compares the development effort of the different controller development approaches.
6. Finally, in Chapter 7 we discuss the results achieved in the thesis and propose future work guidelines for the models and the architecture.

Chapter 2

State of the Art

> Human behavior flows from three main sources: desire, emotion, and knowledge.
>
> Plato

Video games are supported by many technologies, from the purely visual and aesthetic disciplines, such as graphics design, to the extremely technical hardware technology, like GPU massive multi-threading. In this enormous pastiche of mechanisms, technologies and methodologies, an important aspects to be considered are the models that represent the behavior of the characters, as presented in the previous chapter. From these models we observed some technologies that are not fully applied in the video game development and we think that should be applied and that they would make a significant improvement on the results of the final product and even in the process itself.

In this chapter we review the current advances in the fields of machine learning and of cognitive psychology that can be applied on video game environment. Moreover, these fields are so wide that they enclose a large number of technologies and models. We will center our review on some particular models from these fields that we considered as promising for the application on these specific developments.

2.1 Cognitive Psychology Models of Emotions

The quantitative analysis of human emotions is a topic widely treated in psychology. From biology approaches to cognitive science, within the domain of all of these approaches, there are a variety of ideas applied to projects that try to create a synthetic emotional framework in order to evaluate and/or to simulate the emotional response of humans or agents. As Scherer et al. said in their review work [77]:

> *Affective computing is a growing concern both in industry and science. Industry hopes to render technologies such as robotic systems, avatars in service-*

related human computer interaction, e-learning, game characters, or companion devices more marketable by endowing the "soulless" robots or agents (which one might reasonably gloss as "lacking affect") with the ability to recognize and adjust to the users' feelings as well as to send appropriate emotional signals.

The synthetic model for emotions came from the necessity of formalizing the theoretical assumptions described in formal psychology. Psychological theories of emotion have typically been cast at an abstract level and through informal (natural language) descriptions. Concepts in the theory are usually not defined formally, and how processes work may not be laid out in systematic detail. The formulation of a computational model enforces more detail.

As computational modeling exposes hidden assumptions of the theory, addressing those assumptions can extend the scope of the theorizing. Thereby, computational models become not only a way to specify theories, but also a framework for theory construction. Thus, computational modeling also extends the language of emotion theorizing by incorporating concepts, processes, and metaphors drawn from computation, as much as concepts such as how information processing and symbol systems impacted psychology in general. For example, several computational models have recast the appraisal theory in terms of concepts drawn from AI, including knowledge representation, planning [21, 31], neural networks [76], BDI agents [69] and decision–making [34].

The complete family of theories and models applied to computational models of emotions is large, but there are a set of widely used theories for the creation of these model. The main theoretical fields used are the Appraisal theory and the Dimensional theory but many others are also proposed (anatomical and rational approaches), a figure that can illustrate this vast family is extracted for the Scherer's work [77] (see Figure 2.1).

In order to illustrate the different models used in this thesis, in this Section, we briefly introduce the different theories that are used as basis in the models studied during this work. Hereafter, we explain the different basic psychology theories that mainly support the principal branches and models of emotions (Appraisal and Dimensional Theories). In the next section, we present some models that are used as a baseline for the emotional model researches proposed in this thesis. First we review the EMA models (Section 2.2.1) and FearNot! (Section 2.2.2) as models purely derived from Appraisal theories, and then we review the ALMA (Section 2.2.3) and WASABI models (Section 2.2.4) as models that present some principles from the two main branches, mixing the Appraisal and the Dimensional theories.

2.1.1 Appraisal Theory

The predominantly applied theory to emotional models is the Appraisal Theory. In Appraisal Theory, emotion is argued to arise from patterns of individual judgment concerning the relationship between the events and the individual's beliefs, desires and intentions,

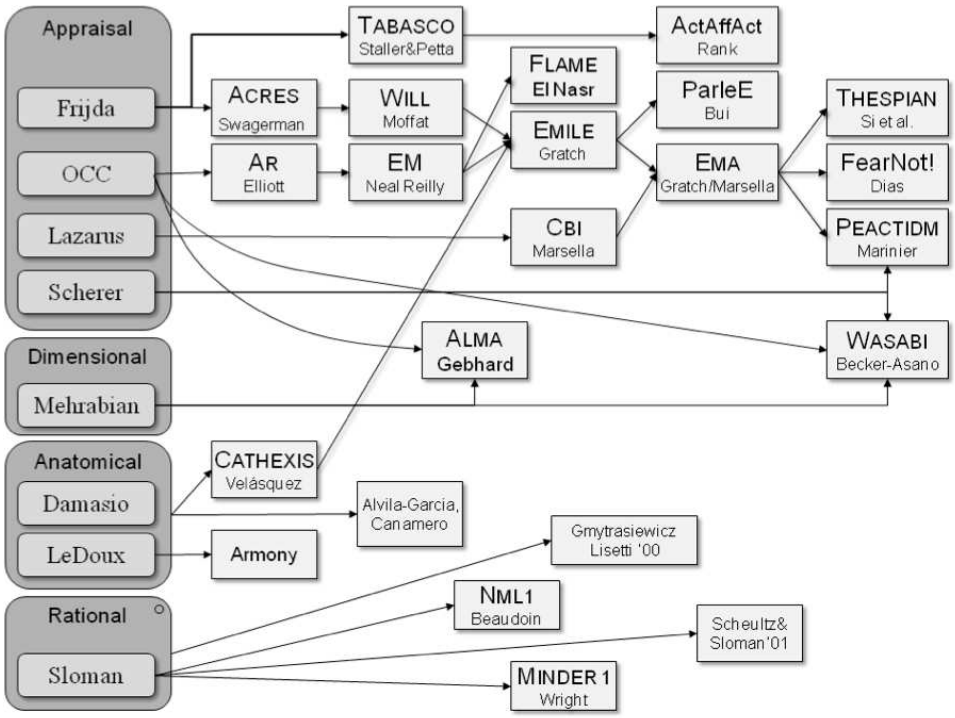


Figure 2.1: A history of computational model of emotions[77]

sometimes referred to as the person-environment relationship [39]. These judgments, formalized through reference to devices, such as situational meaning structures or appraisal variables [26], characterize aspects of the personal significance of events.

In terms of the underlying components of an emotion, the Appraisal Theory foregrounds appraisal as a central process. Appraisal theorists typically view appraisal as the cause of emotion, or at least of the physiological, behavioral and cognitive changes associated with emotion. Some appraisal theorists emphasize "emotion" as a discrete component within their theories, whereas others treat the term emotion more broadly to refer to some configuration of appraisals, bodily responses and subjective experience (see Ellsworth's work [23] for a discussion).

Some theories arise from this assumption. Roseman's appraisal theory [73] treats the emotional events as motive accordance or motive non-accordance. This distinction allows the agent to evaluate the goal alignment with the event. Therefore, it provides mechanisms to estimate the emotion produced by this event. The model proposed by Scherer and Ekman [78] makes use of appraisal as an information processing system, and by means of this, the model examines changes in the present emotional status based on five subsystems. Besides, the OCC model [60] estimates emotions as derived from three different sources: (1) the consequences, (2) the actions of the agents and (3) the evaluations

of the objects.

Recently, the scientific community has experienced a significant expansion in research on computational models of human emotional processes, driven both by their potential for basic research on emotion and cognition as well as their promise for an ever-increasing range of applications.

In general, appraisal theories of emotion usually present a complex architecture with well-founded principles of psychology and robust pillars of cognitive models, such as the EMA model [46], WASABI [7] or FearNot! [21].

2.1.2 OCC Appraisal Theory

The OCC model was proposed on 1988 by Ortony, Clore and Collins in their Cognitive Model of Emotions [60], now it is treated as a standard model for emotion synthesis. A large set of applications uses the OCC model to simulate or analyze emotion in artificial characters [4, 22, 21]. A synthetic model for emotions is needed in order to achieve more believability for embodied characters or even for artificial characters that represent human-like actors. Characters need a model so they can understand the emotions and express them. The emotional model enables artificial characters to argue about emotions as a human actor does. A given event, perceived by an agent, that could produce on a human a specific emotion, as losing money, must raise the same emotion on the agent. According to the OCC model, every situation that a character may encounter can be evaluated and this model, indeed, has a structure of variables, such as likelihood of an event or the familiarity of an object, that can quantify the intensity of the emotion.

In the Cognitive Model of Emotions each of the emotions described has three main elements:

1. The type specification provides, in a concise sentence, the conditions that elicit an emotion of the type in question.

2. A list of tokens is provided, showing which emotion words can be classified as belonging to the emotion type in question. For example, “fright”, “scared”, and “terrified” are all types of fear. (Of course, “fear” is also a type of fear.)

3. For each emotion type, a list of variables affecting intensity is provided. These variables are local to the emotion type in question, i.e., global variables (such as arousal) that affect all emotions are not included. The idea is that higher values for these variables result in higher emotional intensities.

The OCC model presents 22 hierarchic emotional categories based on balanced reactions to situations constructed either as being goal-relevant events, as it acts of an accountable agent (including itself), or as attractive or unattractive objects. The hierarchy contains three branches, namely emotions concerning consequences of events (e.g., joy and pity), actions of agents (e.g., pride and reproach), and aspects of objects (e.g., love and

hate). Additionally, some branches combine to form a group of compound emotions, namely emotions concerning consequences of events caused by actions of agents (e.g., gratitude and anger). Because these notions (i.e. events, actions, and objects) are also commonly used in agent models, this makes the OCC model suitable for use in artificial agents.

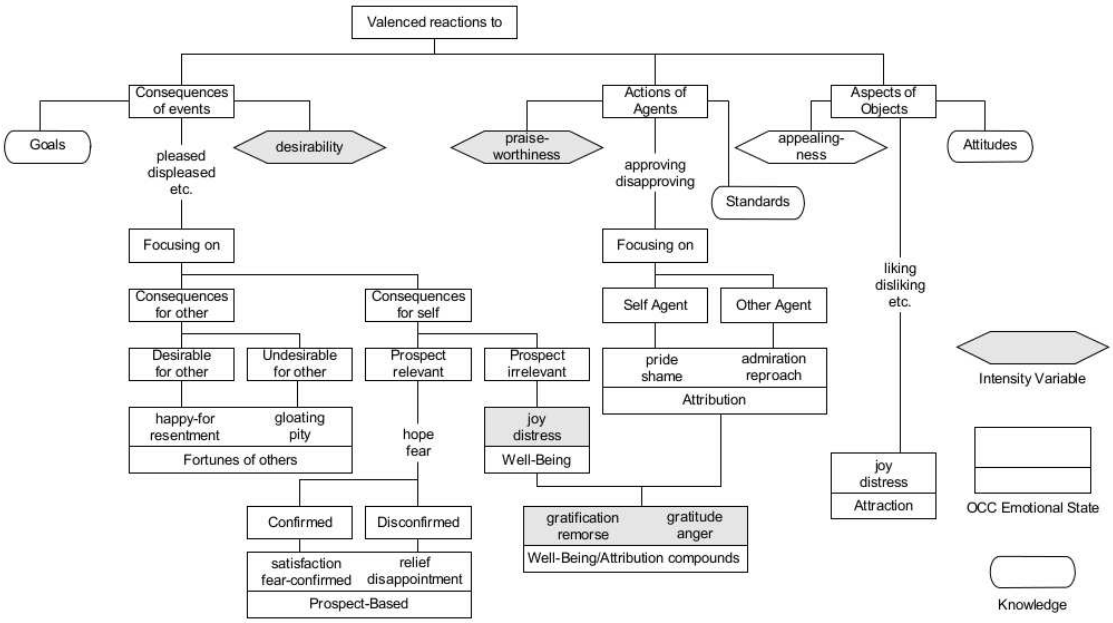


Figure 2.2: Event appraisal on OCC Model[4]

This model also offers a structure for the variables, such as likelihood of an event or the familiarity of an object, which determines the intensity of the emotion types. It contains a sufficient level of complexity and detail to cover most situations an emotional character might have to deal with.

The analytic decomposition of the events to produce the corresponding emotions is based on the branch selection according to the aspect to evaluate at each time (see Figure 2.2), for instance, the consequence analysis of an event can produce utterly the emotions related to "The Fortune of Other" emotions. To identify the correct emotions the OCC Model bases its mechanism in a set of variables that represent the corresponding relation of the subject with his environment and the relationships that he has. Following the same example, the consequences are related with the goals pursued by the subject of the analysis (the desirability of the consequences aligned to the goals of the target), but it is also important to estimate the relationships with the target of the consequence of the event (i.e, friend, foe or oneself).

Bartneck et al. [6] discuss the complexity of the OCC model, to clarify the process that characters follow from the initial categorization of an event to the resulting behavior of the character. They suggest that it can be split into phases:

1. Classification: In the classification phase the character evaluates an event, action or

object, resulting in information on what emotional categories are affected.

2. Quantification: In the quantification phase, the character calculates the intensities of the affected emotional categories.

3. Mapping: The OCC model distinguishes 22 emotional categories. These need to be mapped to a possibly lower number of different emotional expressions.

Classification on OCC

In the classification phase an event, action or object is evaluated by the character, which results in information on what emotional categories are affected. This categorization requires the character to know the relation of a particular object, for example, to its attitudes. Depending on this evaluation either the "love" or "hate" emotional category will be affected by the object.

Quantification on OCC

The intensity of an emotional category is defined separately for events, actions and objects. The intensity of the emotional categories resulting from an event is defined as the desirability and for actions and objects praiseworthiness and appealing respectively (see Figure 2.2). One of the variables that is necessary to calculate desirability is the hierarchy of the character's goals.

Mapping on OCC

If the emotion model has more categories than the character has abilities to express them, the emotional categories need to be mapped to the available expressions. This is very domain dependent, but in many cases it is not necessary to identify or distinguish between all the 22 categories. It is interesting to observe that the emotions produced along the appraisal evaluation are clearly identified in each category and it is only a matter of simplification that the mapping reduces the categories to a subset.

2.1.3 Dimensional Theory

Dimensional theories of emotion argue that emotion and other affective phenomena should be conceptualized, not as discrete entities but as points in a continuous (typically two or three) dimensional space [75, 52]. These theories de-emphasize the concept of the discrete categorization of the emotion, arguing that the label of the emotions and emotional states are error prone and tend to relegate the classification to a label that can be identified as certain body state. Rather the Dimensional theories emphasize concepts such as *mood*, affect or more core affect. Thus, a person is said to be in exactly one affective state at any moment [75] and the space of possible core affective states is characterized in terms of

broad, continuous dimensions. The most widely used representation for these continuous dimensions is the PAD three dimensional model proposed by Mehrabian and Russell [52], where these dimensions correspond to *Pleasure* (a measure of valence), *Arousal* (indicating the level of affective activation) and *Dominance* (a measure of power or control).

Models influenced by dimensional theories, not surprisingly, emphasize processes associated with core affect; other components (e.g., appraisal) tend to be less elaborately developed. Core affect is typically represented as a continuous time–varying process that is represented at a given period of time by a point in 3D space that is"pushed around" by eliciting events. Computational dimensional models often have detailed mechanisms that explain how this point changes over time (e.g., decay to some resting state) and incorporates the impact of dispositional tendencies such as personality or temperament [29].

PAD Dimensional Emotional & Temperament Models

The works on the use of the three parameters for classifying, measuring and applying emotions and temperament are presented by Mehrabian in [49] and [51]. These two works are fundamental for the representation of continuous space that can handle the emotion and moods of an agent, in these works is proposed a framework for the representation of emotional states and temperament of a person. Therefore, the emotions can be represented in a three dimensional space and this space we can also present more stable and lasting emotional states that they are called *moods*.

The models (as well as, the scales and space associated to its framework) were designed specifically to address substrate of connotative and metaphorical meanings which were viewed as being essentially emotion-based. The PAD model presents three orthogonal scales of emotions: *pleasure-displeasure* ($\pm P$ component, representing affective states) as the emotional counterpart of positive-negative evaluations, *arousal-nonarousal* ($\pm A$ component, i.e., mental and/or physical activity) as the correlate of stimulus activity, and *dominance-submissiveness* ($\pm D$ component, as, for example, the control over the situations) as the negative correlate of stimulus potency. Mehrabian postulates that every emotional state predisposes a person toward certain select sets of behavior. For example:

- a more pleasant emotional state is more conducive to a person acting in a friendly and sociable manner with others; conversely, an unpleasant emotional state (e.g., a headache) tends to heighten chances that the individual will be unfriendly, inconsiderate, or even rude with others

- very high arousal is detrimental to concentration on complex tasks and is likely to result in avoidance of, or poor performance on, such tasks

- a more pleasant and, for most tasks, a less aroused emotional state enhances one's desire to work (so, if your office is uncomfortably cold or hot, you are less likely to want to do the jobs that await your attention)

- sexual desire increases with pleasure, particularly when pleasure is combined with greater arousal and/or dominance; sexual dysfunction (e.g., impotence, inability to experience orgasm) becomes more probable when the individual is experiencing displeasure, particularly with displeasure is combined with greater arousal and/or submissiveness

- of all emotional states (i.e., all possible combinations of high vs. low P, A, and D), the pattern of -P-A-D or boredom is the most conducive to nagging desire for food and overeating.

Behaviors, in turn, influence how a person feels. For instance, eating is generally arousing and pleasant and these emotional effects vary depending on the complexity, variety, and quality of the food consumed. Repetitious physical behavior (e.g., rocking in a chair, pacing back and forth in a limited space, walking, jogging, rowing) tends to reduce arousal and increase pleasure and dominance. Thus, repetitious activities are common among non-medicated schizophrenics or autistic children who experience uncontrollably high arousal states. Exercise is also an excellent way in which anyone can counteract anxious or depressed mood. Because exercise reduces arousal, it is even more beneficial in counteracting anxiety than it is in relieving depression.

In general, from the standpoint of simulating human behavior, it is far more important to focus on emotion effects on behavior than on behavior effects on emotion. This is because changing situations throughout the day have continuously powerful effects on emotions and thereby influence action. Action, in turn, tends to reflect the emotional substrate and, in this way, the behavior-emotion linkage is strengthened. There are exceptions, as when a bored person eats (an action that increases pleasure and arousal and thus temporarily counteracts the underlying emotion). So, early AI models can probably perform a reasonably satisfactory simulation by simply focusing on the following chain: Effect of situations on emotions and then the impact of emotions on behavior.

The models proposed by Mehrabian are used in two different processes of the affect evaluation. Both models are extremely general and both uses the PAD dimensions. The first model evaluates the emotions and the second one the temperament, namely the short and long term affect influence mapping, which enables the regular representation of these two aspects of the cognitive psychology.

The PAD Emotion Model

The PAD Emotion Model[49] is a system for the measurement and description of emotions. Three basic dimensions of emotion are used: Pleasure-Displeasure ($\pm P$) or estimation of the liking or disliking, Arousal-Nonarousal ($\pm A$) or general level of physical activity and mental alertness, and Dominance-Submissiveness ($\pm D$) or feelings of control vs. lack of control over one's activities and surroundings. $+P$ and $-P$ can be used as shorthand

notations for pleasure and displeasure; $+A$ and $-A$ represent arousal and nonarousal; $+D$ and $-D$ represent dominance and submissiveness, respectively.

Thus, for instance, anger is represented by very low pleasure ($-P$), high arousal ($+A$), and high dominance ($+D$). Fear, in contrast, is represented as $-P+A-D$. It is seen that anger differs mainly from fear, because anger involves greater feelings of dominance (or control) than fear. More precisely,

$$\text{Anger} : (-0.51, 0.59, 0.25) \qquad \text{Fear} : (-0.64, 0.60, -0.43)$$

showing that fear involves even less pleasure than anger, about the same level of arousal as anger, and considerably less dominance than anger. Any emotion term can be described similarly using the three PAD dimensions. According to the 22 emotional tags of the OCC model, the complete projection of the emotions into the PAD space can be done as shown in the Gebhard's model [29]. The descriptions are of course obtained using mini experiments in which participants are given a single emotion term and are asked to describe that feeling using the PAD Emotion Scales. Anywhere from 20 to 50 individuals independently rate a single feeling. Averages of their ratings on the P, A, and D dimensions yield the coefficients in equations such as the two for anger and fear.

Thus, based on the moderately positive correlation between dignity and anger, the likelihood that a dignified robot will act in an angry way would be moderately high, whereas, using the negative correlation between sadness and anger, the likelihood that a sad robot will act in an angry way would be quite low.

PAD Temperament Model

The PAD Temperament Model[51] is a very general descriptive system for the study of temperament and personality. The model is based on same three dimensions of the PAD Emotion Model (*P-A-D*).

Temperament is distinguished from emotional states in that it refers to an individual's stable or lasting emotional characteristics (i.e., emotional traits or emotional predispositions). More precisely, temperament is an average of a person's emotional states across a representative variety of life situations. A set of three PAD temperament scales has been developed and shown to provide a reasonably general description of emotional traits or temperament.

The three basic dimensions of temperament in the PAD Model are Trait Pleasure-displeasure, Trait Arousability, and Trait Dominance-submissiveness. The relative predominance, across situations and over time, of a person's positive affective states over negative ones defines that person's Trait Pleasure-displeasure. Trait Arousability refers to the strength of an individual's arousal reactions to high-information (i.e., unusual, complex, or changing) situations. Stated somewhat imprecisely, but simply, arousability indexes the strength of a person's emotional reactions to both positive and negative situations. Trait

Dominance is defined in terms of characteristic feelings of control and influence over one's affairs and surroundings versus typical feelings of being influenced and controlled by situations and others.

The PAD temperament space can be conceptually and semantically divided into eight different octants according to the different sign of the different components (view Table 2.1). This division is important in order to fully understand the different representations of the variety of emotions and moods that a character could experience.

+P+A+D	Exuberant	**-P+A+D**	Hostil
+P+A-D	Dependent	**-P+A-D**	Anxious
+P-A+D	Relaxed	**-P-A+D**	Disdainful
+P-A-D	Docile	**-P-A-D**	Bored

Table 2.1: PAD Space Octants

2.2 Emotional Models for Computational Agents

2.2.1 EMA: A Process Model of Appraisal Dynamics

As we said previously, the model proposed by Marsella and Gratch [46] represents an emotional model driven by the appraisal theory. In it, the emotions are considered as inherently dynamic, linked to the world's dynamics and the dynamics of the individual's physiological, cognitive and behavioral process. Appraisal theories posit that the emotion arises from the person's interpretation of his relationship with the environment.

The multi-level theories of appraisal [54, 80] propose that the appraisal process conflates *appraisal* (fast, memory-based process of association) and *inference* (slow, deliberative process). On the other hand, the EMA model argues that the appraisal and inference are different processes that operates over the same mental representation of the person's relationship with the environment. Inference is sequential and slow, but the Appraisal is fast, parallel and automatic. Differences in the temporal course of the emotion dynamics are accordingly due to the differences in the temporal course of the perceptual and inferential processes that constructs the representation of the person-environment relationship.

The EMA model is a fast, single-level of appraisal that can flexibly utilize the output of variety of perceptual and inferential cognitive processes, some slow and deliberative and some fast and automatic. The appraisal dynamics are essentially dictated by the time course of whatever cognitive processes are involved in interpreting and responding to an event: Appraisal results evolve as cognitive processes update the agent-environment relationship.

To fully address the question of the processes that underlie appraisal, we must go beyond such abstract descriptions to detail the processes by which the values of the different appraisal variables are determined. Additionally, the basic mapping from the appraisal to emotions of specific type, intensity and durations must be specified. Completing the

cycle, the impact of the emotions on the coping responses and subsequent changes in the environment-person relationship must be detailed.

Theoretical Basis

The EMA model gets the basic considerations about the emotional theories:

- Appraisal is a process of interpreting a person's relationship with their environment.
- The interpretation can be characterized in terms of a set of criteria (appraisal dimensions, variables or checks).
- The specific emotions are associated with certain configurations of these criteria.
- Certain inferences are minimally necessary to distinguish between emotions [81]:
  - *Relevance, valence and intensity*: emotions are associated with the detection and assessment of events of personal significance. In computational models they must represent events, actions and their immediate consequences, as well as the valence and intensity of these consequences to the agent.
  - *Future Implications*: Some emotions are about the events to come (hope or fear) or are reactions to expectation violations (surprise, disappointment). A computational model must represent future goals and expectations and must include mechanisms for assessing the likelihood of events and actions and their consequences. Interactions between possible outcomes must also be included.
  - *Blame and responsibility*: Appraisal theories assume that a first step for represent a response to an emotion-evoking event is, usually, on the cause and the source agent responsible for its occurrence. Unlike causal reasoning, appraisal theories argue that causal attribution and responsibility are key factors on the emotional prompting, also the intentions of the source and/or the consequences for thirds are important for the appraisal of emotions. The computational models of appraisal must withstand the notions of causality and agency, as well as other actor's motivations.
  - *Power and coping potential*: An important factor in people's emotional response is their subjective sense of control over emotion-eliciting event. To reason about individual power, a computational model of emotion must therefore represent the social power of the agents, the coercive relationships between agents such organizations or hierarchies, the external power of the individuals (over the world and other agents) and the internal power of the agents (the ability to abandon a cherished goal or overturn a preconception). To reason about adaptability and to support so-called *emotion-focused* coping strategies, the models must be open to subjective reinterpretation.

– *Coping strategies*: Patterns of appraisal elicit emotional behavior, but they can also trigger cognitive responses referred as *coping strategies*. These cognitive responses act changing the environment or person's representation (e.g. plans, beliefs, desires and/or intentions). These includes problem-focused strategies (plans, etc.) and emotion-focused motivations (state changing, etc.). A computational model of emotions must provide mechanisms for translating the patterns of appraisal into appropriate external actions or changes of configurations such as beliefs, desires, intentions and plans.

EMA Computational Model Assumptions

The basic theory ambiguities are resolved in the EMA model with the following assumptions.

Appraisal causes emotions: As Frijda referred as *law of situated meaning*[26], the appraisal of events in the environment causes the emotional responses on the agents and allow the incidental influence of emotional states through the simple notion of mood.

Cycle of appraisal and re-appraisal: The person's coping response is central to explain the dynamics of appraisal and emotional responses [39]. The EMA model assumes a cyclical relationship between appraisal, coping and re-appraisal. A person's initial appraisals if a situation provokes a variety of cognitive and behavior responses that change the person's relationship to the environment. The resulting cycle of appraisal and re-appraisal is a central element in explaining the dynamics of emotion.

Appraisal is shallow and quick: The EMA model proposes a clean distinction between inference (the cognitive processes) and appraisal (simple evaluations, reactive, parallel and unique). The appraisal, as a single-level process, is fast and parallel with other processes (some slow, some fast) that perform inferences over the representation of the person-environment relationship.

The EMA model assumes that a representation of the "agent-environment relationship" is continuously update. Furthermore, the represented agent-environment relationship is appraised, continuously and automatically, resulting in emotional and coping responses. The actions of the agent also change the environment (and his relationship with it), both action and inference are influenced by the coping responses. In addition, the world also change dynamically without agent intervention, due to other agents taking actions, as well as natural events and processes.

The model explicitly represents intermediate knowledge state, that may be appraised, augmented by further inference, and transformed by coping responses. The representation of this knowledge states and facilitates fast appraisals.

*EM*otion and *A*daptation Model

To support the rapid and sequential unfolding of emotional responses EMA uses a representation built on the causal representations developed for decision-theoretic planning, augmented by the explicit representation of intentions and beliefs. Planning representations capture a number of essential distinctions required for computing appraisal, such as causal reasoning, detect future benefits and threats, etc. The decision-theoretic notions of probability and utility allow EMA to compute the appraisals of desirability and likelihood. Explicit representations of beliefs and intentions allow to distinguish merely contemplated actions from those an agent is committed to perform, an important distinction for computing attributions of blame and responsibility.

The agent's interpretation of its "agent-environment relationship" (called *causal interpretation*) can be seen as corresponding to the content of the agent's working memory and provides a uniform, explicit representation of the agent's beliefs, desires, intentions, plans and probabilities that, in turn, allows uniform, fast appraisal processes. At any point in time, the *causal interpretation* represents the agent's current view of the agent-environment relationship which changes further observation or inference.

Knowledge Representation

The agent-environment relationship is a mixture of symbolic and numeric representations. The causal interpretation is organized into: (1) record of past events, (2) current world state and (3) possible future outcomes.

1. *States and actions*: EMA represents the state of the world as a conjunction of propositions. Actions are represented with preconditions and effects. Actions are assumed to have a duration and their effects can occur asynchronously, so at a given time several actions can be executing simultaneously.

2. *Beliefs and intentions*: States and actions are annotated with epistemic variables representing the beliefs, desires and intentions of agents in the situation. Belief correspond to commitments to the truth value of propositions and are binary (true/false) although probabilities represent a measure of the certainty in this commitment. The model allows the agent to distinguish between act intention and outcome-intention. This allows the model to represent unintended effects of actions. The model also represents probabilities over these intentions to represent uncertainty when inferring another agent's intentions or uncertainty in another agent's ability or willingness to fulfill public commitments.

3. *Causal relations*: The causal interpretation represents several relationships between actions and states.

4. *Establishment relations* represent that an effect of some action establishes a precondition of some other actions. *Threat relations* represent that the effect of some actions

blocks the precondition of another action.

5. *Probabilities and Utilities*: Utilities represent agents' preferences for states. Probabilities over states represents the agent's certainty in the truth-value of the state at some point in time. Probabilities over actions are of two forms. P_I represents the likelihood that an agent intends to execute an action; P_E represents the probability that an action can be executed.

Cognitive Operators

EMA is built on SOAR architecture [58] which presents the operators correspondence to deliberative processes that are posited to be relatively slow and sequential; and reactive processes are posited to be fast, automatic and parallel. EMA organizes mental processes around a set of primitive cognitive operators (Table 2.2) that utilize and update the current causal interpretation. Some but not all cognitive operators change the contents of the causal interpretation.

Cognitive	Update belief
	Update Intention
	Update plan
	Understand speech
	Output speech
	Wait
Perceptual	Monitor goal
	Monitor expected effect
	Monitor expected act
	Listen to speaker
	Expect speech
	Monitor unexpected event
Motor	Initiate action
	Terminate action

Table 2.2: Cognitive Operators of EMA

Appraisals

The EMA model assumes that the appraisal is fast, parallel and automatic. This is achieved by modeling appraisal as a set of continuously active features, and the detectors that map these features of the causal interpretation are modeled into appraisal variables. In this sense, appraisal do not change the causal interpretation but provides a continuously updated "affective summary" of its content.

EMA appraises each and every proposition that is represented in the causal interpretation. The model associates a data structure, called *appraisal frame*, with each proposition. This appraisal frame maintains a continuously updated set of appraisal variables (Table 2.3) associated with each proposition.

Relevance	Derived from the utility of a state for the agent or the utility of a state causally derived from the current state	*Perspective*	The viewpoint from which the proposition is analyzed. Usually (or only used) is the agent's own viewpoint.
Desirability	Positive or negative value of the proposition to the agent whose perspective is being taken.	*Likelihood*	The measure of the likelihood of outcomes
Expectedness	This the extents to which the truth value of a state could have been predicted from the causal interpretation.	*Causal Attribution*	Who deserves credit / blame for executing an actions
Controllability	Measure of the possibility that the outcome can be altered by actions under control of the agent.	*Changeability*	The extent to which an event will change of its own accord.

Table 2.3: Appraisal variables of EMA

Emotions, Mood and Focus of Attentions

The EMA model supports a two-level notion of emotions state - appraisal and mood - that can account for some of the indirect effects of emotion documented in empirical research. The appraisal level determines the agent's coping response but this is biased by an overall mood state. The EMA theoretical perspective on mood is that the initial appraisal of a situation leads to recruitment of brain and bodily resources that facilitate certain mental and physical activities and thereby change the subsequent appraisal of the situation. However, EMA does not explicitly model such bodily consequences of appraisal.

The appraisal level maintains multiple appraisal frames (one for each proposition in causal interpretation) each of which is labeled with a specific emotion type and intensity, and each competing to determine the agent's coping response. At the mood level, individual frames are also aggregated into a higher-level *mood*. This aggregate frame: (1) is a summary

of various appraised events; (2) dissociated form the original eliciting event and (3) which tends to change slowly over time as appraised frames are added or removed in response to changes in the causal interpretation.

The Mood State representation is made by a set of emotion labels (e.g. Hope, Fear, ...) with an [0..1] intensity that is a function of all appraisal frames with the corresponding type. The mood state has an indirect effect on appraisal in that EMA applies a mood adjustment to individual appraisal frames depending of the type of mood related.

EMA's moment-to-moment coping response is determined by a simple activation-based focus of attention model that incorporates both appraisal and mood. Specifically, the appraisal frame that determines the coping is the most recently accessed appraisal frame with the highest mood-adjusted intensity.

Coping Strategies

Coping determines, moment-to-moment, how the agent responds to the appraised significance of events. Within EMA, coping strategies are proposed to maintain desirable or overturn undesirable in-focus events (appraisal instances). Coping strategies essentially work in the reverse direction of the appraisal that motivates them, by identifying features of the causal interpretation that produced that appraisal and that should be maintained or altered.

The EMA coping strategies can be classified as referred at Table 2.4

2.2.2 FearNot!

The architecture built by Dias and Paiva [21] was also an application of the Appraisal Theories inspired on the OCC model. The proposed agent architecture aims to create an autonomous agent believable and empathic, inspired by some of the elements present on the traditional animation:

- **Believability and Empathy**: The characters must be believable and be able to produce empathic reactions with users.

- **Reactive and Cognitive Capacities**: Believable characters should display motivations, goals and desires, which is only possible if they have cognitive capacities. Also, the character should react as quickly as necessary in a rapidly changing environment.

- **User Interaction**: The character should be able to interact with an external user and receive suggestions.

- **Generality**: The agent architecture should be domain independent. It must allow the easy creation of different characters with different personalities for different domains.

Related To	Strategy	Description
Attention	Seek Information	Form a positive intention to monitor the pending, unexpected or uncertain state that produced the appraisal frame
	Suppress Information	Form a negative intention to monitor pending, unexpected or uncertain state that produced the appraisal frame
Belief	Shift Responsibility	Shift an attribution of blame/credit from (toward) the self and toward (from) some other agent
	Wishful Thinking	Increase / Lower the probability of a pending desirable / undesirable outcome or assume some intervening act or actor will improve desirability.
Desire	Distance / Mental Disengagement	Lower utility attributed to a desired but threatened state.
	Positive Reinterpretation	Increase utility of a positive side-effect of some action with a negative outcome.
Intention	Planning / Action Selection	Form an intention to perform some external action that improves an appraised negative outcome.
	Seek Instrumental Support	Form an intention to get someone else to perform an external action that changes the agent-environment relationship.
	Make Amends	Form an intention to redress a wrong.
	Procrastination	Defer an intention to some time in the future.
	Resignation	Drop an intention to achieve a desired state.
	Avoidance	Take an action that attempts to remove the agent from a looming threat.

Table 2.4: Coping Strategies in EMA

Emotion and Dynamics of Emotion

The concept of emotion of this work steams from OCC cognitive theory of emotions. The OCC emotion type represents a family of related emotions differing in terms of their intensity and manifestation, for example, it references the possible set of emotions resulting from appraising the prospect of a goal expect to fail, with a varying degrees of intensity: concerned, frightened, petrified ... The prospected model considers the following attributes for the description of an emotion:

- **Type**: The type of the emotion being experimented.
- **Valence**: Denotes the basic types of emotional response. Positive or negative value

of reaction.

- **Target**: The agent targeted by the emotion.
- **Cause**: The event/action that causes the emotion.
- **Intensity**: The intensity of the emotion.
- **Time-Stamp**: The moment in time when the emotion was created or updated.

Every emotion has associated an *Intensity* attribute which is assigned with different values depending on the different situation that generated the particular emotion. The emotion does not remain constant during its life cycle in the system, the emotion must be attenuated through time in order to reflect the dynamics of the emotional system itself. This characteristic reflects the notion that an emotion does not last forever and does not affect the evaluation of the subsequent emotional states in the same way. The model uses a decay function for emotions proposed by Picard [72]:

$$Intensity(em, t) = Intensity(em, t_0) \times e^{-b \cdot t} \tag{2.1}$$

Mood and Arousal

In addition to emotions, the proposed model represents arousal and mood:

- *Arousal* represents the degree of excitement of the character, this model only uses the psychological arousal. Aroused characters will feel intense emotions. Every time that a character experiences a high intensity emotion his arousal level will rise. Therefore, if nothing new happens, the character will "calm down".
- *Mood* represents an overall valence of the character's emotional state and is also used to influence the intensity of emotions. The idea, based on Picard, is that characters with bad mood will tend to experience more negative emotions, and characters with good mood will experience more positive emotions. Mood is represented as an internal variable that increases when positive emotions are created and decreased with negative emotions. This variable also decays over time until it reaches its neutral value.

Personality

The character's personality is based in OCC and is defined by: (1) a set of goals, (2) a set of emotional reaction rules, (3) the character's action tendencies, (4) emotional thresholds and (5) decay rates for each of the 22 emotion types defined in OCC. This model uses two of OCC goal types, active-pursuit goals and interest goals. Active-pursuit goals are goals that the characters actively try to achieve. Interest goals represent goals that a character has but does not pursue, like avoiding to be hurt. The emotional reaction rules assess how generic events are appraised and represent the character's standards and

attitudes. Action tendencies represent the character's impulsive actions as action tendencies is due by Lazarus [39], which states that action tendencies are innate biological impulses, while coping is "a much more complex, deliberate and often planed psychological process". OCC model specified for each emotion type an emotional threshold and decay rate. The emotional threshold represents the resistance of the character towards an emotion type. The decay rate specifies how fast the emotion decays over time.

Architecture

The model presents two layers for appraisal coping processes. The reactive layer is responsible for the character's action tendencies, while the deliberative layer achieves the agent planned behavior. Action tendencies represent hardwired reactions to emotions and events that are rapidly triggered and performed, these tendencies depend on the character's emotional state, so it can only be made after the appraisal process. The cognitive appraisal depends on the agent's plans and can take some time; when an event is received the continuous planner has to update the active plans, even before the start of the emotional reactions. The deliberative level generates prospect-based emotions (hope, fear, satisfaction,...) based on the agent's plans and goals, the reactive level generates all other types of OCC emotions using a set of domain dependent emotional reaction rules as used by Martinho in S3A [47]. When an event is perceived, the reactive appraisal matches the event against the set of defined emotional rules, generating the corresponding emotions.

Cognitive Appraisal

A continuous planner [75] that uses a partial-ordered-plan builds up the core of the deliberative layer. This planner was extended to include probability information about actions and to perform emotion-focused coping strategies. Each character has defined a set of active-pursuit goals that are triggered upon certain conditions. Thus, every time the agent receives a new perception from the environment, the deliberative layer checks all deactivated goals to determine if any of them become active. If so, an intention to achieve the goal is added to the intention structure. Initial hope and fear emotions based on the goal's importance are created in this process. The OCC theory of emotions does not specify how emotions affect reasoning/cognition and action selection. The model uses the emotions to determine the most relevant intention: the ones generating the strongest emotions are the ones that require the most attention from the agent, and thus are the ones selected by the planner to continue deliberation. After selecting the strongest intention, the best plan built so far is brought into consideration. This process is named focus and generates the following prospect based emotions:

- *Hope of success*: to achieve the intention.
- *Fear of failure*: to achieve the intention.

- *Inter-goal fear*: for not being able to preserve an interest goal, a goal to protect/-maintain.

The final phase of the deliberative appraisal checks all the active goals to determine whether they succeed or fail. When such events occur or if the planner is unable to make a plan, more prospect based emotions will be generated, such as *Satisfaction*, *Disappointment*, *Relief* and *Fear-Confirmed*.

Coping

The coping strategies performed over the selected plan depend on the character's emotional state and personality. This model uses two types of coping: problem focused coping and emotional focused coping. The first one focuses on acting on the environment to cope with the situation, thus it consists on planning a set of actions that achieve the pretended final result and executing them. The second works by changing the agent's interpretation of circumstances, thus lowering strong negative emotions.

2.2.3 ALMA Model

The ALMA Model [29] is based on *Affects*: General term for feelings, emotions, or moods the conscious subjective aspect of feeling. This model creates a hybrid approach between the Appraisal Theory, based on the OCC Model and the Dimensional Theory of Merhabian.

This model is divided into three layers, with time-related decomposition (see Figure 2.3):

1. long-term provided by the personality,
2. middle-term by the moods and,
3. short-term emotional component.

Personality

The long-term layer provided by the personality is based on the **Big Five Norman-Goldberg's theory** [59] centered on five factors and their constituent traits:

- **Openness**: appreciation for art, emotion, adventure, unusual ideas, curiosity, and variety of experience.
- **Conscientiousness**: a tendency to show self-discipline, act dutifully, and aim for achievement; planned rather than spontaneous behavior.
- **Extraversion**: energy, positive emotions, urgency, and the tendency to seek stimulation in the company of others.

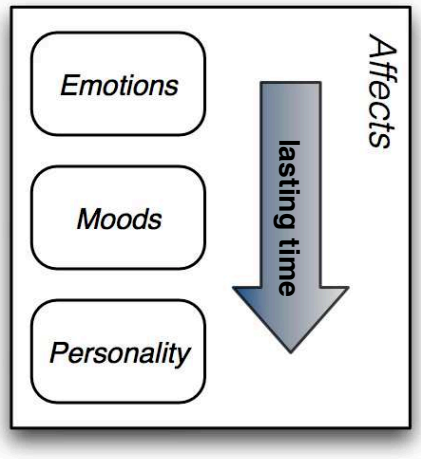


Figure 2.3: ALMA layer model

- **Agreeableness**: a tendency to be compassionate and cooperative rather than suspicious and antagonistic towards others.
- **Neuroticism**: a tendency to experience unpleasant emotions easily, such as anger, anxiety, depression, or vulnerability.

The personality of a character sets the initial value of the mood of the character in a "neutral" state. The personality is not easily changed and, for virtual characters, may be considered constant.

Mood

The middle layer of the ALMA model is based on the *mood* of the character, which it derives from the default mood and the affects that the character has received recently. Moods are stable affective states which have a great influence on human's cognitive functions[57] and they are modeled by the PAD model[51]. According to Mehrabian, every possible human emotion can be represented as definite points in the three dimensions. The following emotions exist as points in that space: angry, bored, curious, dignified, elated, hungry, inhibited, loved, puzzled, sleepy, unconcerned, and violent. Violent, for instance, represents a displeased, highly aroused, and highly dominant emotional state. This layer centers the perception, as said by Gebhard: "*The mood is the base of the perception of the environment of a character.*"

Emotions

The third layer of the ALMA model is defined by the emotions produced by an affect perceived by the character. The emotions reflect short-term affects, which are usually bound to a specific event, action or object. After their elicitation emotions usually decay

and disappear of individual's focus These emotions are cataloged by the OCC Emotional Model (Ortony, Clore y Collins on the 1988)[60]. Each event perceived by a person is cataloged according with: (1) the effects that it has on the target, source and observer of the action, (2) the action itself and (3) by the objects involved in the action and the perception and relevance that it has for the observer. There are many agents based on this model, the emotional characters include COSMO [40], Émile [31], Peedy [2], and the Greta agent [15] although the approaches differ in the granularity of modeling, the mathematical machinery for computing emotions, and in the way of how the model has been implemented on a technical level.

Appraisal Mechanism

Each appraisal of an action, event or object, lets the engine of the ALMA model produce an active emotion. This emotion, after been generated, decays over a certain amount of time. All active emotions are used as input of the *pull and push mood change function*. It first computes the virtual emotion center of all currently active emotions in the PAD space by using a certain mapping.

The pull and push mood change function works in the following way (see Figure 2.4): If the current mood position is between the PAD space zero point and the virtual emotion center, the current mood is attracted towards the virtual emotion center. This is called pull phase. If the current mood is beyond (or at) the virtual emotion center the current mood is pushed away, further into the current mood octant (from octants described in the PAD Temperament Model, Section 2.1.3) in which the mood is located. This is called push phase. The push phase realizes the concept that a person's mood gets more intense the more experiences the person make that are supporting this mood. The intensity of the virtual emotion center defines how strong the current mood is attracted respectively pushed away.

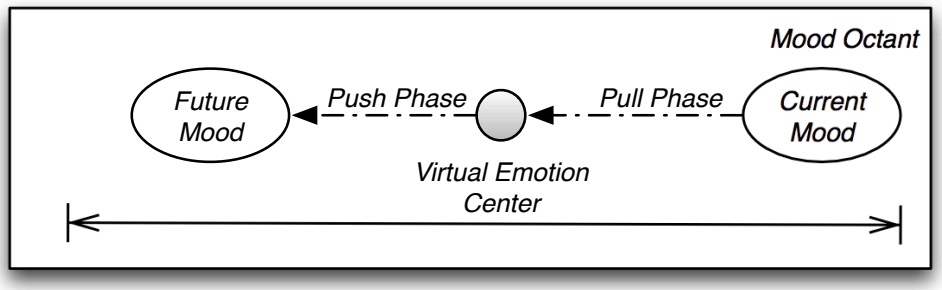


Figure 2.4: ALMA Pull-Push Mood Change Function

2.2.4 WASABI: Affect Simulation for Agents with Believable Interactivity

The WASABI architecture [7] combines bodily emotion dynamics with cognitive appraisal in order to simulate infant-like primary emotions as well as cognitively elaborated (more adult) secondary emotions. The specification works with the concepts of *Emotions* and *Moods*.

- *Emotions* are understood as current states with a specific quality and intensity, which are the outcome of complex neurophysiological processes for communication. These processes include neural activity of the brain as well as physiological responses of the body. One gets aware of one's emotions in two cases: (1) if his awareness likelihood w exceeds a certain threshold or (2) if one concentrates on the underlying processes by means of introspection.

  Emotions can be classified into primary and secondary ones, but every emotion has either positive or negative valence of a certain value and compared to mood an emotion lasts significantly less. Therefore, the WASABI model differentiates the secondary emotions from primary since the secondary emotions are:

  - based on more complex data structures than the primary ones,
  - more dependent of the memory and reasoning process of the agent than from the primary emotions,
  - responsible of the expression and verbalization at embodied agents.

- *Mood* is modeled as a background state with a much simpler affective quality than emotions. In contrast to the model of Gebhard [29], mood is not derived from PAD space, but modeled as an agent's overall feeling of well-being on a bipolar scale of positive versus negative valence already before a mapping into PAD space is achieved. Any non-neutral mood is slowly regulated back to a neutral state of mood much slower than it is the case for emotional valence. Accordingly, a mood's duration is in general longer than that of any emotion.

Primary Emotions

The primary emotions are inborn affective states, which are triggered by reflexes in case of potentially harmful stimuli. Each of the primary emotions is located in PAD space according to the coordinates derived from some of the values given in Russell and Mehrabian[52]. The WASABI architecture describes nine primary emotions, 5 from the six emotions defined by Elkman, and another three, so WASABI has the primary emotions at Table 2.5.

#	Primary Emotion	PAD value
1	Angry	$(80, 80, 100)$
2	Annoyed	$(50, 0, 100)$
3	Bored	$(0, 80, 100)$
4	Concentrated	$(0, 0, \pm 100)$
5	Depressed	$(0, 80, 100)$
6	Fearful	$(80, 80, 100)$
7	Happy	$(80, 80, \pm 100)$ $(50, 0, \pm 100)$
8	Sad	$(50, 0, 100)$
9	Surprised	$(10, 80, \pm 100)$

Table 2.5: Primary emotions at WASABI

Secondary Emotions

According to Damasio [19], the elicitation of secondary emotions involves a "thought process", in which the actual stimulus is evaluated against previously acquired experiences and online generated expectations.

The "prospect-based emotions" cluster of the OCC model of emotions is considered here to belong to the class of secondary emotions, because their appraisal process includes the evaluation of events against experiences and expectations.

This OCC cluster consists of six emotions (namely fear, hope, relief, disappointment, satisfaction, and fears-confirmed), of which *hope*, *fears-confirmed*, and *relief* are simulated in the WASABI architecture.

Emotional Dynamics

The implementation of emotion dynamics is based on the assumption that an organism's natural, homeostatic state is characterized by emotional balance, which accompanies an agent's normal level of cognitive processing. Whenever an emotionally relevant internal or external stimulus is detected, however, its valence component serves as an emotional impulse, which disturbs the homeostasis causing certain levels of Pleasure and Arousal in the emotion module. Furthermore, a dynamic process is started by which these values are continuously driven back to the state of balance.

The two valences are mathematically mapped into PAD space and combined with the actual level of Dominance, which is derived from the situational context in the cognition of the architecture. This process results in a course of a reference point in PAD space representing the continuously changing bodily feeling state from which the awareness likelihoods of primary and secondary emotions are incessantly derived.

WASABI embodiment

There is a conceptual distinction of cognition and embodiment in the WASABI architecture. Any perceived stimulus is appraised by conscious and non-conscious processes in parallel leading to the elicitation of "emotional impulse". These drive the "emotion dynamics", which is part of the agent's virtual embodiment and from which mood, Pleasure, and Arousal are continuously derived. PAD space is used (1) to directly elicit primary emotions with a certain intensity and (2) to act as an "awareness filter", which ensures mood-congruency of both primary and secondary emotions. The resulting set of "aware emotions" is finally reappraised in the cognition before giving rise to deliberative actions.

The conceptual distinction of an agent's simulated embodiment and its cognition is presented and the different modules and components of the WASABI architecture are assigned to the corresponding layer are represented at the Figure 2.5.

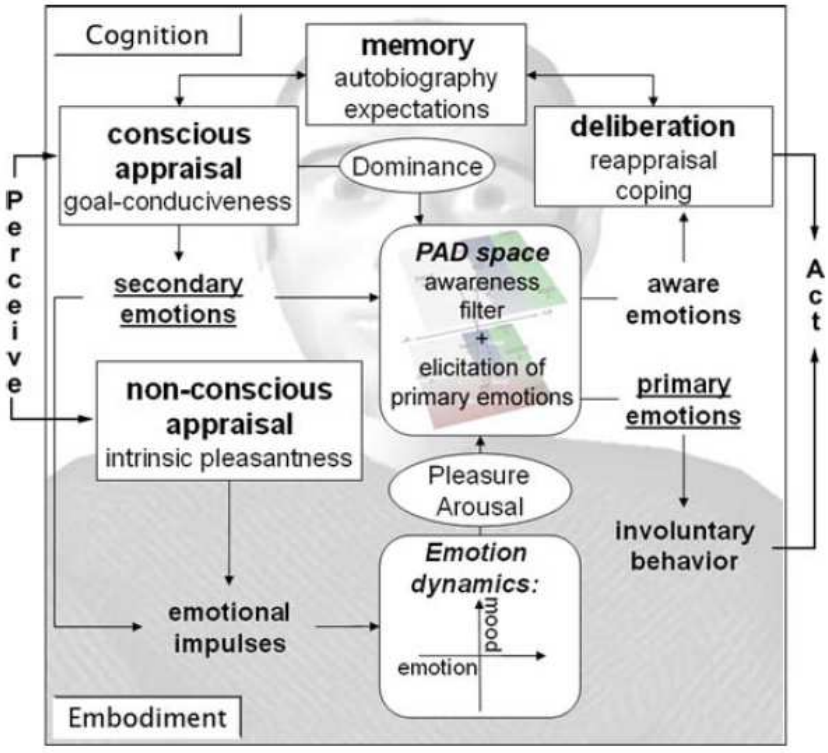


Figure 2.5: WASABI Embodiment Architecture

The cognitive layer of the WASABI architecture is based on a BDI-Agent model which enables the abilities to update his memory and generate expectations. These deliberative processes not only permits WASABI to derive his subjective level of Dominance from the situational and social context, but also propose cognitively plausible secondary emotions.

2.3 Reinforcement Learning Techniques

Reinforcement Learning (RL) is the problem faced by an agent that must learn behavior through trial-and-error interactions with a dynamic environment. Thus, RL is a machine learning technique applied to model the behavior of an agent that has sensor mechanisms to perceive the state of an environment and obtains rewards from the actions it performs within this environment.

There are two main strategies for solving reinforcement-learning problems. The first is to search in the space of behaviors in order to find one that performs well in the environment. This approach has been taken by work in genetic algorithms and genetic programming. The second is to use statistical techniques and dynamic programming methods to estimate the utility of taking actions in states of the world. These techniques take advantage of the special structure of reinforcement-learning problems that is not available in optimization problems in general.

In the standard RL model (see Figure 2.6) the agent is connected to its environment Σ via a perception of its state ε and the actions α that it takes. On each step, the agent receives an input η from the environment and a reward ρ (*reinforcement signal*) from the actual state of the environment derived from the past actions. According to this information the agent's behavior selects an action α which maximizes a long-term measure of reward. The action α changes certain parameters that represent the environment and produces the new state that is passed as input to the agent with the resultant reward.

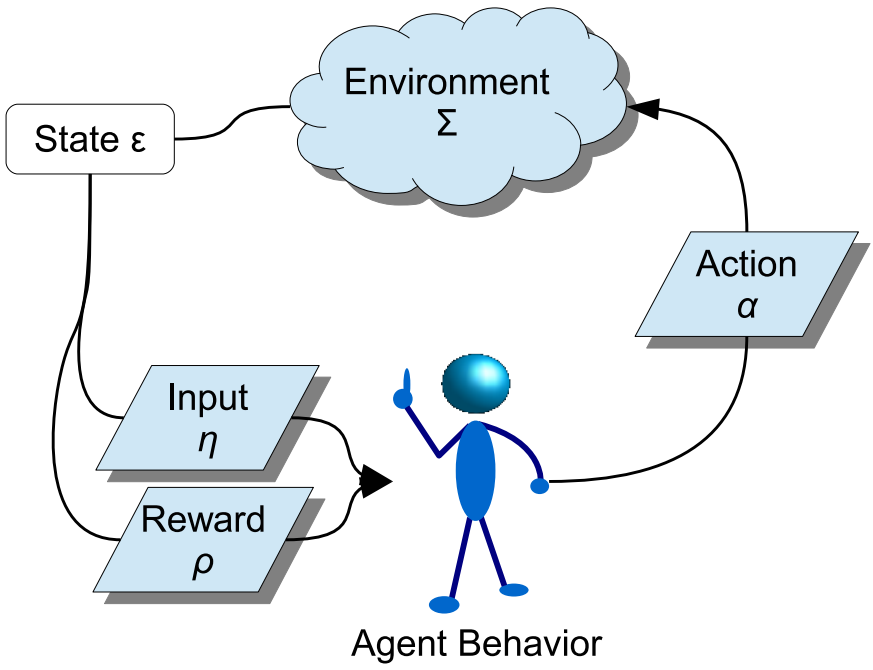


Figure 2.6: A standard reinforcement-learning model

Formally, a RL model consists of:

- a discrete set of environment states Σ,
- a discrete set of action Λ, and,
- a set scalar reinforcement signals Γ , typically real numbers.

Reinforcement learning differs from the more widely studied problem of supervised learning in several ways. The most important difference is that there is no presentation of input/output pairs. Instead, after choosing an action the agent is told the immediate reward and the subsequent state, but not which action would have been in its best long-term interest, so it is necessary for the agent to gather useful experiences about the possible system states, actions, transitions and rewards actively to act optimally. Another difference from supervised learning is that on-line performance is important: the evaluation of the system is often concurrent with learning.

Markov Decision Processes

RL is widely applied to solve a broad range of problems [35]. One of the typical problems that are solved by RL algorithms are Markov Decision Processes (MDPs). MDPs model environments where actions performed by an agent make the state of the environment transition to some other state with a certain probability, making the transitions non-deterministic. An MDP consists of

- a set of states $\mathcal{S}$,
- a set of actions $\mathcal{A}$,
- a reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \longrightarrow \Re$, and
- a state transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$, where a member of $\Pi(\mathcal{S})$ is a probability distribution over the set $\mathcal{S}$ (i.e. it maps states to probabilities). We write $T(s, a, s')$ for the probability of making a transition from state s to state s' using action a.

The state transition function probabilistically specifies the next state of the environment as a function of its current state and the agent's action. The reward function specifies expected instantaneous reward as a function of the current state and action. The model is Markovian if the state transitions are independent of any previous environment states or agent actions.

Although general MDPs may have infinite (even uncountable) states and action spaces, the particular model of the video game scenarios could be represented as a finite state environment with finite action set, so we focus our attention in solving these kind of problems.

Stochastic Games

Moreover, similar cases to the MDPs are the Stochastic Games (SGs) in which multiple agents select actions and the next state and rewards depend on the joint action of all the agents. Thus, the SGs are more natural models for many video games.

The SGs are formally defined by Shapley on early 1950's [79] as:

> In a stochastic game the play proceeds by steps from position to position, according to transition probabilities controlled jointly by the two players.

Similarly to the MDPs, an *n-person* SGs can be decomposed as follows:

1. a finite set $\mathcal{I}$ of players,
2. a finite set $\mathcal{S}$ of states,
3. a finite set $\mathcal{A}^i(z)$ of actions available to player $i \in \mathcal{I}$ at state $z \in \mathcal{S}$, and $\mathcal{A} = \bigcup \mathcal{A}^i$
4. a joint action $a(z) \in \mathcal{A}$, defined as the union of all the actions performed by the n players at state z.
5. a reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{I} \to \Re$; so $r^i(z, a)$ is the reward for the player i in the state z given the joint action a,
6. a state transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$, where a member of $\Pi(\mathcal{S})$ is a probability distribution over the set $\mathcal{S}$ (i.e. it maps states to probabilities). We write $T(s, \overline{a}, s')$ for the probability of making a transition from state s to state s' using a list of actions $\overline{a}$.

This model fits perfectly in the video game scenarios, where the joint actions performed in the environment by the characters produce the transition between states. The real-time simulations or some kind of parallel turn resolution mechanisms implemented in many games propitiates these kind of problems, where the SGs are more suitable.

In the next section, we review some of the classical models used for solving the MDPs and the SGs, as we show in the next chapters of this thesis, the problems that we try to solve are more tractable by the SGs solvers, than from the classical strategies for solving the MDPs. But, we realize how important it is to compare both mechanisms as possible approaches for the resolution of the learning mechanism needed by the proposed architecture.

2.3.1 SARSA and Q-Learning

In the Reinforcement Learning problems, as we defined previously, it is crucial to know how the agent will take the future into account, the reward signals Γ. There are basically three models that try to optimize the reward in different moments. The *finite-horizon* model tries to optimize the reward in the following h steps. The *infinite-horizon*

model considers the attenuated long-term rewards as if they were an interest rate. Finally, the *average-reward* model takes into account the long-term average reward. This delayed reward problems are captured perfectly by the MDPs and SGs, which try to obtain the optimal policy that maximized the expected reward.

In these delayed reward problems, there are two alternatives for obtaining an optimal policy. First, a controller can be learned without learning a model (*Model-free approach*). Second, a model can be learned and, then, a controller can be derived from it (*Model-based approach*). In this thesis more attention will be paid to the *Model-free* approach, due to the complexity of the video game environments that makes difficult the construction of a model of the game.

From the model-free approach, we select two very representative algorithms which are extensively used in the RL problem solving: **SARSA** and **Q-Learning**

SARSA

The SARSA algorithm is an on-policy temporal difference learning algorithm for MDPs [85]. It is based on the estimation of the expected reward of a given state s for a given action a, denoted by $Q(s, a)$ (also know as Q-value).

This estimation is continuously updated according to the following equation:

$$Q(s, a) \Leftarrow (1 - \alpha)Q(s, a) + \alpha \left[r + \gamma Q(s', a')\right] \tag{2.2}$$

where $0 \leq \alpha \leq 1$ is a learning rate parameter (determining how fast the state-action pair is updated) and $0 \leq \gamma < 1$ is the discount factor for future rewards (indicating the influence of the reward from new state-action pair into the reward of the original state-action pair).

The update equation states that for a given state-action pair $(s, a) \in S \times A$ the new state-action value is obtained by adding a small (depending on α) correction to the old value. The correction is the difference between the immediate reward r increased by the discounted future state-action value $\gamma Q(s', a')$ and the old state-action value $Q(s, a)$.

Therefore, SARSA (s, a, r, s', a') is an on-policy learning algorithm in the sense that it estimates the value of the same policy that it is using for control.

Q-Learning

Q-Learning [89] constitutes an off-policy alternative to SARSA, the learned action-value function, Q, directly approximates, Q^* , the optimal action-value function, independent of the policy being followed. Thus, it replaces the term $\gamma Q(s'a')$ by $\gamma \arg\max_{a' \in A(s')} Q(s', a')$ in the above equation 2.2.

This provides a separation of the policy being evaluated from the policy used for control. It leaves the update equation like this:

$$Q(s, a) \Leftarrow (1 - \alpha) \quad Q(s, a) + \alpha \left[r + \gamma \quad \underset{a' \in A(s')}{arg\, max} Q(s', a') \right] \tag{2.3}$$

Action selection

In both algorithms, the process is similar, at each time step, the agent must select an action to perform, to do so, the state in which the environment is, s, is used as entry for the query of a table in which for each action $a \in \mathcal{A}$ and $s \in \mathcal{S}$, we store the expected discounted reward. Intuitively, we select as response the action that has a higher expected reward $arg\,max_{a \in \mathcal{A}}$.

This greedy strategy always exploits the current knowledge and does not explore other possible solutions that can have more expected rewards. There are different strategies to save this problem and they have different approaches for these action selections. From the ϵ-greedy which reserves certain ϵ probability for the exploration of a random action, to more sophisticated approaches of soft-max action selection supported on the Gibbs, or Boltzmann probability distributions, to associate the ϵ probability thought the different actions according to a ranked-weighted order of the actions derived by their expected rewards.

SARSA and Q-Learning in video games

Many Machine Learning algorithms are been applied in the video games environments, as shown the survey of Galway, et al. [27]. Within the ML techniques, the most commonly applied in commercial video games are the Neural Networks. These mechanism have serious drawbacks that make them hard to use extensively, but are a promising approach for certain problems like racing games [86] or neuroevolution in team games [13]. Also, from the different studies about the reinforcement learning in video games, a number of issues have been revealed, including the need for a suitable state-action space abstraction and value function representation.

The application of RL in commercial video games is marginal, and unnoticed. But some scientific researches in these techniques reveal the promising potential in the creation of controllers derived from the automatic learning of strategies. For instance, the RL in fighting video games was introduced by [30], as a possible approach for the automatic creation of fighting strategies; or Madeira, et al. [42] research the application of RL in a turn-based strategy game *Battleground*^{TM} [1].

More recently, Amato and Shani [1] explore the high-level reinforcement learning applied to the Firaxis video game Civilization IV$^{\mathrm{TM}}$.

These studies show that learning agents found interesting policies capturing the behavior of the opponents, evaluating the strategy according to specific reward functions. In this thesis we use the RL algorithms (SARSA and Q-Learning) as starting point for the exploration of different techniques that can be more suitable for this kind of scenarios. These two algorithms fit perfectly with the MDPs problems, but, as it has been said before, the video games scenarios can also be seen and modeled as Stochastic Games. In the next

[1] Talon Soft: http://www.matrixgames.com/products/319/details/John.Tiller%27s.Battleground.Civil.War

section we will introduce a base algorithm that is used for the SGs approach, the Win or Learn Fast algortihm.

2.3.2 Win or Learn Fast. WoLF

Although SARSA and Q-Learning are suitable approaches to deal with MDPs, video games do not fully match MDP characteristics, as mentioned previously.

It is important, in many cases, to represent the complete set of transitions that may occur derived from the presence of two or more agents over the same environment. Stochastic games (SGs) are a natural multi-agent extension of MDPs, and have also been studied within RL [18, 32].

Despite RL algorithms (such as Q-Learning) being appropriate to deal with MDPs, they are less appropriate, in theory, for SGs, due to the multi-agent aspects [10]. Thus, some variants of these algorithms have been successfully applied in these SG scenarios. "Policy Hill Climbing" (PHC) and "Win or Learn Fast" (WoLF) [11] are extensions to the Q-Learning algorithm particularly designed to deal with stochastic scenarios with multiple agents i.e. with SGs.

Both PHC and WoLF maintain a learning rate in the form of a selection probability for each state-action pair. The main difference is that, in PHC, this learning rate is constant while WoLF changes this learning rate depending on whether it is winning or losing. Intuitively, the algorithm tries to learn quickly when it is losing and more slowly when it is winning. To determine whether the algorithm is winning or losing, the current policy's payoff is compared with that of the average policy over time.

In addition to Q-values, the algorithm also maintains the current mixed policy ($\pi(s, a)$). This policy controls the probability of selecting a given action during the learning phase. It is updated by increasing the probability of selecting the best performing action according to a learning rate δ_l, that is applied when the algorithm is losing, and δ_w, that is used when the algorithm is winning, with $\delta_l > \delta_w$. Algorithm 1 provides a detailed description of this policy.

2.3.3 Evolutionary Techniques for Reinforcement Learning

The evolutionary techniques are used for the iterative refinement of solution which explores a set of possible solutions trying to find those which are better (or more promising) solution for the problem. Thus, as Stanley observes [82], it can be used as a complementary technique that handles an underlying Machine Learning algorithm as the problem which we try to optimize, whether it is used for parameter tuning or for structural exploration.

Indeed, the combination of evolutionary strategies and reinforcement learning has mainly been addressed towards the use of optimization algorithms to adjust connection weights in neural networks. There are only few references on the use of evolutionary techniques to complement reinforcement learning algorithms based on Q-Learning or similar

Algorithm 1: WoLF Algorithm

begin

Let α, $\delta_l > \delta_w$ be learning rates, Initialize $Q(s,a) \leftarrow 0, \quad \pi(s,a) \leftarrow \frac{1}{|A|}, \quad C(s) \leftarrow 0$

while *Finalization state not reached* **do**

From state s select action a with probability $\pi(s,a)$

Q-values are updated observing reward r and next state s',

$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$

Update estimate of average policy, $\bar{\pi}$,

$C(s) \leftarrow C(s) + 1 \quad \forall\ a' \in A, \bar{\pi}(s,a') \leftarrow \bar{\pi}(s,a') + \frac{1}{C(s)}(\pi(s,a') - \bar{\pi}(s,a'))$

Update $\pi(s,a)$ and constrain it to a legal probability distribution

$$\pi(s,a) + \begin{cases} \delta & if\ a = \underset{a'}{argmax}(Q(s,a')) \\ \frac{-\delta}{|A|-1} & \text{otherwise} \end{cases}$$

where,

$$\delta = \begin{cases} \delta_w & if \sum_a \pi(s,a)Q(s,a) > \sum_a \bar{\pi}(s,a)Q(s,a) \\ \delta_l & \text{otherwise} \end{cases}$$

end

end

approaches (usually named policy-space approaches). A first reference appeared in Moriarty's work [56]. In this work, a preliminary evolutionary algorithm for reinforcement learning (named EARL) is put forward. EARL evolves a chromosome with the same size as the number of states, and the value of each of the genes would be the action to perform at this state. EARL focuses on deterministic policies.

More related with the video game scenarios, a representative approach was the NeuroEvolution of Augmenting Topologies (NEAT) [83], that have attracted great interest in the video game community. A practical example was the video game called NeuroEvolving Robotic Operatives (NERO) that extends NEAT to work in real-time.

Whiteson and Stone [90] evolved neural networks using NEAT combined with Q-Learning algorithms to search function approximations. Using neural networks in this way is rather different to the approach we adopt in this thesis, though it would be an interesting future work to compare the two approaches. This approach, compared to the one presented in this thesis, also used Lamarkian evolution to evolve individuals that learn, but Q-Learning is only applied to implement an ϵ-greedy selection mechanism. With this same objective, reinforcement learning had been used to guide evolutionary processes in general optimization problems, for example in the participation selection for hybrid dynamic evolutionary algorithms [38].

More recently, Yoshikawa et al. [92] introduce a reinforcement learning algorithm with a hierarchical evolutionary mechanism to evolve adaptive action value tables. The

algorithm evolves several Q-Learning parameters, such as state discretization data, learning rate α, discount factor γ and searching rate (non deterministic action selection).

Another study of evolutionary algorithms applied to artificial intelligence in games is treated by Yannakakis [91], presenting different GA algorithms in a set of testing environments, close related with well-known games. The human evaluation of the general interest of the controllers created by these techniques is analyzed with interesting results. This analysis withstood the applications of GA to the building of interesting controllers for agents in video games.

In the field of robot control, reinforcement learning and evolutionary algorithms have been combined to speed-up the learning process. The combination deals with real-time on-line constrains and high dimensional control problems, as reported by Maravall et al. [43].

A usual combination of evolutionary techniques and learning algorithms are the learning classifier systems (LCS) [87]. LCS model use evolutionary algorithm to evolve if-then rules, called classifiers. LCS is a different approach not considered for this work.

In our approach we propose to combine evolutionary algorithms and reinforcement learning according to the following techniques:

- Evolutionary Algorithms:
  - Estimation Distribution Algorithms (introduced in Section 2.3.3).
  - Differential Evolution (introduced in Section 2.3.3).
- Time Difference Algorithms:
  - SARSA and Q-Learning (introduced in Section 2.3.1).
  - Win or Learn Fast (introduced in Section 2.3.2).

Estimation of Distribution Algorithms

Estimation of Distribution Algorithms (EDAs) were introduced in the 90s [53]. In general terms, EDAs are similar to Genetic Algorithms, but their main characteristic is the use of probabilistic models to extract information from the current population (instead of using crossover or mutation operators) in order to create a new and presumably better population.

As in the case of other evolutionary algorithms, EDAs create multiple solutions or individuals in a population. This population evolves from one generation to the next by estimating the probability distribution of a set of individuals (usually the best individual from the past generation), then sampling the induced model (without using crossover or mutation operators).

Based on the probabilistic model considered, three main groups of EDAs can be distinguished: univariate models, which assume that variables are marginally independent;

bivariate models, which accept dependencies between pairs of variables; and multivariate models, in which there is no limitation on the number of dependencies.

The complexity of the different EDA approaches is usually related to the probabilistic model used, and the ability of that model to identify and represent the (in)dependencies among the variables. Detailed information about the main characteristics of EDAs, as well as the different algorithms that belong to this family, can be found in the work of Larrañaga and Lozano [37].

In this thesis, we use the Univariate Marginal Distribution Algorithm for Gaussian Models (UMDAg) [36]. This algorithm considers no dependencies between the variables involved in the problem. It is assumed that the joint density function follows an n-dimensional normal distribution, which is factorized by a product of one-dimensional and independent normal densities.

Differential Evolution Algorithms

Differential Evolution (DEs) algorithms were proposed by Rainer Storn and Kenneth Price in 1995 [84]. DEs are also a specific type of evolutionary algorithms that use an alternative recombination operator. Given a population in the generation i, each individual in this population $x_{i;j}$ is selected for recombination. The selected vector receives the name of objective vector. Three other vectors, x_{r_1}, x_{r_2}, and x_{r_3} are then randomly selected. They are all different to the objective vector and different to one another. These four vectors are then combined to obtain a new vector candidate to replace the objective vector:

$$v_{i+1;j} = x_{r_1} + F(x_{r_2} - x_{r_3}) \tag{2.4}$$

First, vectors x_{r_2} and x_{r_3} are subtracted and scaled according to a F factor. Finally, the result vector from the previous step and vector x_{r_1} are added. The final result vector of the mutation phases is known as the donor vector.

Once the donor vector is obtained, it is combined with the original vector $x_{i;j}$ by means of a crossover operation. A usual crossover method is the binomial crossover, which randomly selects, component by component, either from the original vector or from the donor vector, producing the new individual $u_{i+1;j}$. Finally, $x_{i;j}$ is replaced by $u_{i+1;j}$ if and only if the new individual has a better fitness value.

A more detailed survey of DE can be found at the work of Das and Suganthan [20].

2.4 Discussion

The application of the technologies reviewed in this chapter to the video game development is hard in the actual state. The reinforcement learning algorithms are hard to include in the production process, and the emotional engines are not suitable for the video game environment. Thus, we observe that the creation of a complete architecture

that can deal with these two components is interesting for the final objective of this work. The creation of this architecture must include the key features of these two fields, and the characteristics involved in the video game environment itself.

We analyze the models presented in this chapter in order to obtain a global vision of the common factors in all of them and, more important, the lack of the features that are relevant for the development of the architecture and the models within it.

Emotional Model Components

The appraisal theory based models provide a complete mechanism for handling emotional responses, the inclusion of any of these models in the architecture of the video game characters is too complex. The video game scenario needs "coarse-grained" emotional states and simple and efficient transitions. On top of that, the design of multiple characters is an extra effort that requires to consider models somehow easier to calibrate and evaluate.

In this discussion we introduce the parameters or elements that are relevant for us in the analysis of emotional models .

Perception / World Representation

The emotional models must represent the world in which the character is living and perceiving, the elements of the environment that the character perceive may or may not produce some emotions in him, but it is important to identify the sources or triggers of the emotions and the parameters that have influence in the variables of the emotion perceived.

Appraisal / Automatic Emotion Analysis

Once an element, action or consequence is perceived by a character, it usually produces some emotional response in the character giving some internal or external procedure that changes the relationship of the agent with his environment. This emotional analysis of the perceived world usually goes apart of the cognitive procedure but must have some influence in it, at least as an additional parameter for the cognitive deliberation of the state of the world.

Emotions in the Cognitive Procedure

The cognitive procedure of any character should be focused in the production of some action in order to achieve certain goal desirable for him, even in the case of secondary characters in a scene, they must produce some actions that are perceived by the primary characters (or human actors) as goal directed even roaming around. The inclusion of emotion in the cognitive engine of a character must add the emotional component to the goals of the character to produce some actions emotionally coherent altering the goal orientation of the agent motivation.

Basic Character Model for Emotions

Not all the characters must react in the same way to the same changes of the state of the world, so we must have some *a priori* representation of the character, his goals, his relations and his behaviors in the emotional context.

Emotion Dynamics / Timescale for Emotions

The emotional stimulus that a character feels in a given instant dissipates along the time and this dynamic must be captured by an emotional model. The emotional dynamics must take the emotion parameters for one point to another of the full spectrum of emotions making believable those emotions and their non-static behavior.

Mood and Affect Representation

The Mood State and the affect must have an internal representation that has to be maintained along the time enabling the addition of more emotions and the dynamic attenuation of the previous emotions. The representation must be rich enough to distinguish between positive and negative affect states. These values representing the current state must also influence the analysis of the future events.

Soft Computing in Video Games

As some studies show and many interviews with the lead programmers of the industry reveal, the soft computing techniques are not implanted in the actual work flow of the video game development. The promising techniques, that are successfully applied in the resolution of many theoretical and industrial problems, are not used by the video game programmers. We think it is derived from the specific knowledge needed to implement them and the lack of total control or flexibility of the solution that they produce, but, we also think that with the correct tools and mechanisms it is possible to introduce these techniques in the future development.

Evolutionary Algorithms in Video Games

Traditionally, Evolutionary Algorithms (EAs) and Evolutionary Computation (EC) are defined as stochastic search methods that mimic the metaphor of natural biological evolution. Evolutionary algorithms operate on a population of potential solutions applying the principle of survival of the fittest to produce better and better approximations to a solution. At each generation, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the problem domain and breeding them together using operators borrowed from natural genetics. This process leads to the evolution of populations of individuals that are better suited to their environment than the

individuals that they were created from, just as in natural adaptation. EAs are particularly useful on problems with little domain knowledge because they require do not a priori analysis of the hypothesis space or of the problem domain.

As Lucas and Kendall describe [41]:

> *Evolutionary Computation is well identified as a game (the game of life) in which their participants may try to propagate its genetic material across the generations and the fitness value of an individual depends on the friends and foes interaction as well as to the environment. This process is known as Co-Evolution in the Evolutionary Computation literature and becomes of the evolution of a game strategy without need of a human interaction or expertise.*

The application of Evolutionary Algorithms in the commercial video games has been relegated to some niche games that use the evolutionary background as part of the game itself, like Spore™[2] or Creatures™[3]. These video games illustrate with the artificial life paradigm the features underlying the EAs. On the other hand, the video game developers are reluctant to applying these techniques to their projects but, from the academic research there are promising works that must be taken into account for further developments like the Yannakakis' [91] work that uses the EAs for the objective measure of *interestingness* to evolve more engaging ghost behaviors for a simplified version of Pac-Man and other prey and predator games. Also, the works based on NEAT algorithm that produced new types of games like the NEROGame[4]. These researches keep the door open for the application of EA mechanisms in the future.

In this thesis, the EAs are used for the creation of a hybrid algorithm that makes an improvement in the implementation of the RL techniques for the automatic creation of controllers that should be an important point for the improvement of the playing experience through the development of more creative, sophisticated and/or challenging AI controllers for the characters of the next generation video games.

Machine Learning in Video Games

The are many techniques applied in research scenarios closely related with the video games, like the simulators, and the serious games. In the last few years more research groups are beginning to work in the video game scenarios as part of their applied development. In the case of the machine learning many works derived from the neural networks or hierarchical planners are being tested in frameworks that are basically video games. The multi-agent technologies are also turning their focus to these environments that can be easily modeled and handled.

[2] http://www.spore.com

[3] http://www.gamewaredevelopment.co.uk/creatures_index

[4] http://nerogame.org

Moreover, there is still a gap between the necessities of the video game industry and the research developments, that is why we think that an architecture that can be easily implanted in existing video game engines can be helpful for the future projects, and can approach the machine learning paradigm to the entertainment industry in a structured way.

In this thesis we try to make a practical approach to the mechanism, processes and algorithms that are used in order to achieve the objective of creating an useful set of technologies included in an architecture that can improve the AI applied in the creation of the environment, characters or plots of this important industry.

Chapter 3

General Model Architecture

> Architecture is basically a container of something. I hope they will enjoy not so much the teacup, but the tea.
>
> Yoshio Taniguchi

As we discussed in the Section 2.4, there are different techniques that can be applied to improve the development of the video games. Some of them are related to the automation of the construction of controllers (for the growing amount of characters in the upcoming games), while other methods or models address the design of more appealing and entertaining worlds bringing more sense of realism (through behavior and environment enrichment).

The creation of more sophisticated controllers for video game characters is the key to the new generation of products that are likely to appear in the next years. The support for the creation of these controllers can be conducted in many directions. In this thesis, the improvement in the creation of the video game character controllers, is to be achieved through the application of two different objectives:

- First, we focus our attention on **emotion and affect simulation** in the environments present in the video game scenarios. For the credibility of the behaviors, it is of foremost importance the correct interpretation of the events produced in the environment and the affective reaction, not only to a particular event, but also to a sequence of events. Thus, the actions and effects perceived in the environment by a character drive his emotional state to a certain point. Therefore, the behavior of this character must show this mood throughout his actions.

- Second, the **automatic creation of certain controllers** for the video game characters will be supported. The complexity of the manually created controllers, and the believability of the behaviors generated by them, is constrained by the work force dedicated to the development and debugging of the components of the intelligence

in charge of the behavior of the characters. Hence, the automatic creation of these components can produce a broader range of controllers that can be created by the exploration and exploitation of the environment state.

The application of these two objectives to the construction and development of the behaviors of characters can lead to the automatic creation of video game character controllers. These automatically created controllers are not only economically competitive compared with the hand-coded alternatives, but also richer (with more varieties in their strategies) due to automatically-discovered behaviors based on their current emotional state. This emotional state is produced by the perception of the changes and events prompted in the environment, and the analysis of these events that the character does based on his own personality.

In this chapter, we present the general architecture designed for the automatic creation of emotional behaviors. The components that are part of this architecture are abstractly presented in the following sections. We conceive the complete model as a modular structure that admits different implementations of its components as long as these implementations fulfill certain requirements and interfaces.

The chapter is organized as follows: in Section 3.1 we present the proposed architecture that will enclose the two objectives aforementioned, in Section 3.2 we describe the structural composition of the architecture. We explain the communication among the components in the dynamic view of the architecture in Section 3.3, in Section 3.4 we introduce some design and implementation guidelines to support the application of the architecture to a eventual development. And finally, in Section 3.5, we discuss the architecture as contribution in the context of video game development.

3.1 General Description of AGCBAR Architecture

The model presents an architecture for the creation of game controllers, called AGCBAR (Automatic Game Controllers Behaviors with Affect Responses) Architecture. This architecture encloses the necessary components to automatically create and exploit character controllers. The characters under the control of the AGCBAR Architecture are supposed to display enriched and interesting behaviors (in terms of believability and playability). These controllers take the environment changes not only as an observation of the world that produces some reactive responses on the character according to his current strategy, but also, they take these environment changes as events that can prompt some emotional responses. These emotional responses can change or modify the general behavior and/or objectives of the characters activating different strategies, which are selected based on the emotional state rather than on pure logic.

3.1.1 AGCBAR Architecture Concerns

The AGCBAR Architecture provides two key features that support the design of the video game character controllers. These concerns are:

1. *Mood Dynamics*, which represent the interpretation and processing of the events prompted in the environment to produce emotional values. The dynamic treatment of the emotions (produced by the events announced by the game engine) creates a certain "flow" of the character's mood across different states. This flow can be used to modify the behavior of the character according to these sequences of events. It does not mean that the behavior will be 100% emotional, indeed there is a strategic planning in the control mechanisms, but the emotional state dictates the objectives of this strategy. It is an emotion-driven strategy planning (ranging from more elaborate control plans to pure reactive responses).

2. *Automatic Controller Creation* involves the procedure of creating different strategies according to the observed states of the environment variables produced by the game engine. These strategies are different combinations of the actions available for the characters to interact with the environment presented by the game engine. As mentioned before, the creation of these controllers pursues a given set of different goals, which could be defined as a pure strategic goal (showing intelligent behavior) or any other different goal driven by the emotional states (showing a broad range of flavors of affective behaviors).

These features are mainly implemented by two components: (1) the **Emotional Engine** that must capture and parse the events of the environment to produce the *Mood Dynamics* and (2) the **Learning Engine** that must build the strategies and behaviors according to the environment observations, in order to implement the *Automatic Controller Creation*.

This architecture is designed to be applied to a broad spectrum of game engines with a minimal interference with them, so it can be attached in a "clean way" to the game engines. Furthermore, the two components that form this architecture are designed so they can work in the absence of any of them. This makes the architecture able to be tailored according to the necessities of any given video game scenario. Thus, we can use the Automatic Controller Creation without the Mood Dynamics and vice versa.

3.2 AGCBAR Architecture. Structural View

The AGCBAR Architecture, as can be seen in Figure 3.1, comprises a set of main components.

There are three different engines described in the architecture, which will be fully detailed in Section 3.2.1:

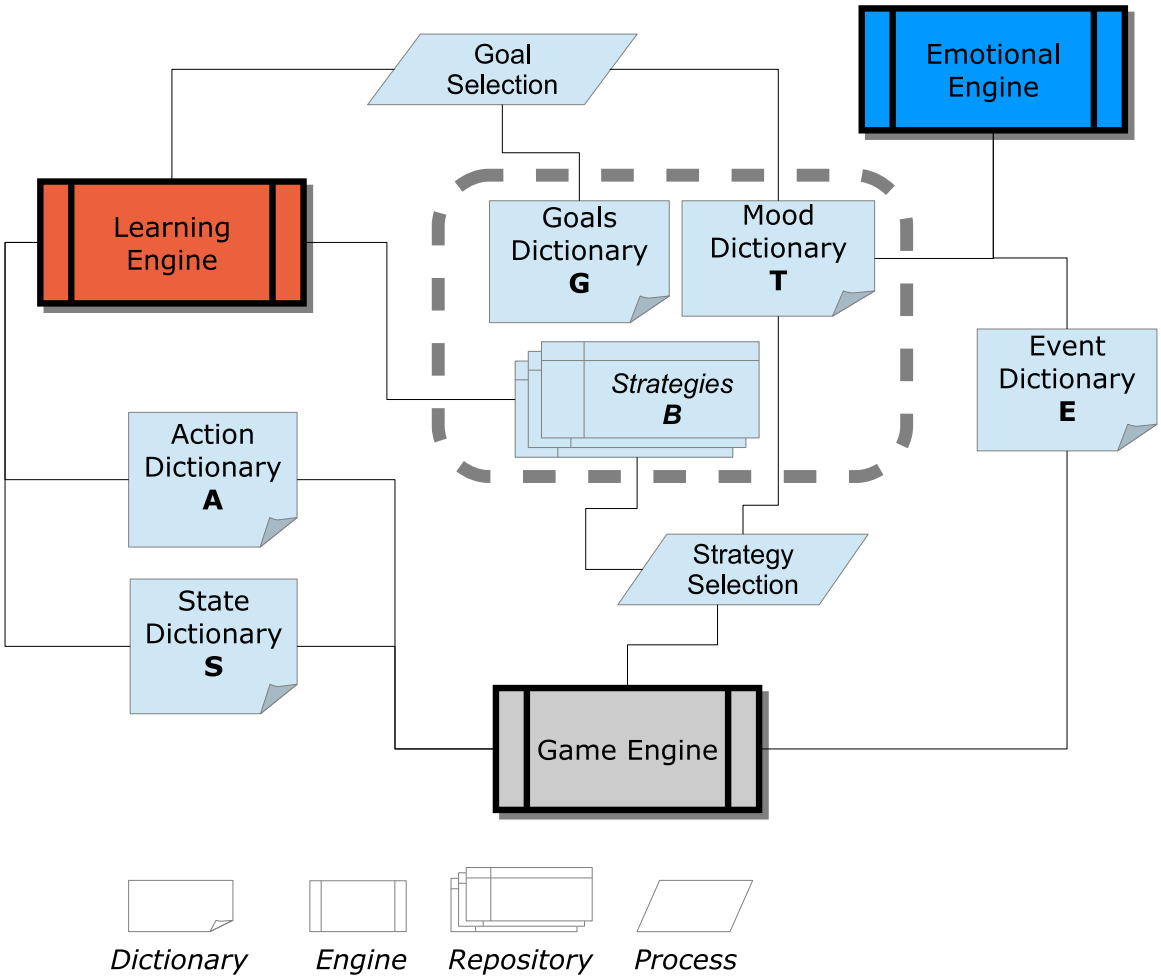


Figure 3.1: AGCBAR Architecture: Structural View. The relationship among the Goals, Moods and Strategies enables the features addressed by the architecture.

- **Game Engine**: this is the base component that manages the game logic and rules. It encloses the simulated environments and the character controller interfaces. The Game Engine interacts also with the human player through the inputs that the player provides to the system (actions steered by the human player user interface) and displaying the state of the environment by the rendering of the scene in a display or screen.

- **Emotional Engine**: this engine is the component in charge of the *Mood Dynamics* concern, which will produce changes in the emotional state of the characters according to the events produced by the environment.

- **Learning Engine**: the *Automatic Controller Creation* is handled by this component. It produces different behaviors (Strategies) according to the changes in the

environment and adapt them to the different goals that the character can have.

Aside of these components, there are two processes, which are secondary processing units that drive the behaviors to use or explore. Although these processes could be implemented as a very complex procedures, for the purpose of this thesis we have selected a simple version of them, just to focus our attention on the aforementioned Engines.

- **Strategy Selection**: if the *Mood Dynamics* concern is applied in a development, the current Mood, provided by the Emotional Engine, will be used as part of the Strategy Selection process. This process will choose the strategy (among those created by the Learning Engine during the Automatic Controller Creation Phase, for example) to be used until the next change in the character's mood.
- **Goal Selection**: during the phases enclosed in the *Automatic Controller Creation* concern, the Learning Engine has to create a strategy that tries to maximize the performance, when given a certain goal. In the AGCBAR Architecture, the goals are related to the mood state that the character has in a precise instant, thus the Learning Engine has to create as many strategies as the number of character's moods.

The components' functionality, interaction and description are based on a central definition of concepts that all the architecture shares. These concepts are described in a common component called **Architectural Dictionaries**

AGCBAR Architecture: Architectural Dictionaries

These Architectural Dictionaries store the different values and elements that the different components are going to use or exchange. In the Figure 3.2, we present the different dictionaries and repositories that the designer must complete, and the components related to them (as producers or consumers of them).

In order to specify these dictionaries, the designer must identify the following elements:

1. Event Classes [*Events Dictionary*]: The events, $E_i \in \mathbf{E}$ are emotionally relevant events produced by the Game Engine which can cause some changes in the emotional state of a character. They are described as a class with the relevant attributes to represent these events parametrized. These emotional events are used by the Emotional Engine during the Mood Dynamics application. For instance, some relevant events could be: the death of a character in the scene, or the perception of certain effect (an explosion, a suspicious sound, etc.), so they can produce a particular emotional response in the characters that perceive them.
2. Mood Tags [*Moods Dictionary*]: The emotional states of the character, called Moods, must be represented with a finite set of values, $T_i \in \mathbf{T}$. The Emotional Engine will

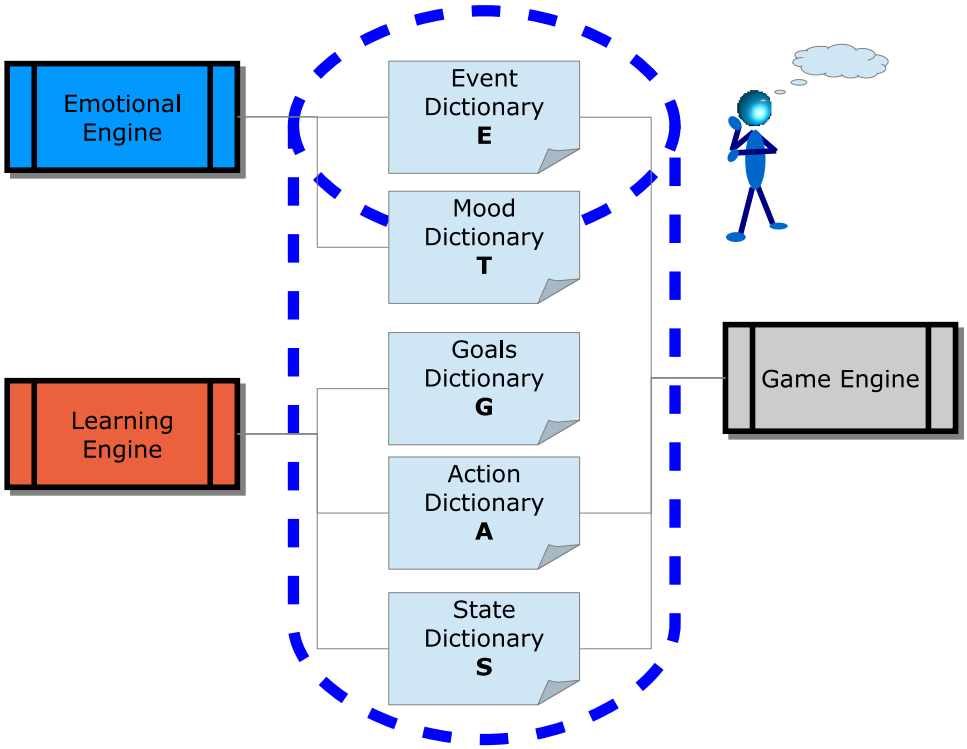


Figure 3.2: AGCBAR Architecture: Dictionaries. The different dictionaries are related to the engines that compose the AGCBAR Architecture.

select which of them should be used at each instant during the Mood Dynamics stage. This mood discriminates which of the strategies (behaviors) the character uses to decide his actions. For example, a Mood could be tagged as `Afraid` to represent the state where the character is worried and a bit scared about the situation.

3. Goals [*Goals Dictionary*]: In the AGCBAR Architecture, the goals, $G_i \in \mathbf{G}$, are preferences about the different states that the game scenario can have. These goals are closely related to the scenario presented to the character in the video game. They are usually described by a concept related to the game, such as "survive the maximum time possible" or "minimize the damage received".

4. Actions [*Actions Dictionary*]: The actions described in this dictionary, $A_i \in \mathbf{A}$, are the actions that the Learning Engine uses for strategy construction. They can be the same actions that are implemented in the Game Engine as atomic actions for the characters, but also, they can be more abstract actions that describe complex action sequences or actions with some parameters that the Game Engine has to interpret when they are selected as result of a certain strategy. For instance, the Action Dictionary can include an action "Move Forward" which will represent a particular action in each instant in the Game Engine according to the actual facing direction of the character,

or "Get First Aid Kit" which can enclose actions to navigate to the nearest medical kit and grab it.

5. States [*States Dictionary*]: The environment state, simulated on the Game Engine, must be represented in some precise way in order to extract a strategy that is appropriate for a given state. These representation of the state can be described by the designer in this dictionary. The states $\overrightarrow{S_i} \in \mathbf{S}$ are used by the Learning Engine to establish the current state in which the character must select and action or, the state we want to evaluate according to a certain goal. Therefore, the state of the environment in a specific instant is described by a set of variables that takes a particular value in this instant. In the states dictionary we describe which are those variables and their possible values.

Moreover, the common factors of the two processes of the AGCBAR Architecture are the production (in the case of the Goal Selection Process) and selection (in the Strategy Selection Process) of the Strategies, $B_i \in \mathbf{B}$, which will be stored in the *Strategies Repository*. A strategy, B_i, is defined as a probability distribution of actions $A_i \in \mathbf{A}$ for each state $\vec{S}_i \in \mathbf{S}$. Therefore, in the AGCBAR Architecture, to fulfill the two concerns (*Mood Dynamics* and *Automatic Controller Creation*), **the correct alignment among the Moods, Goals and Strategies (behaviors) is necessary**, see Figure 3.3. Even in the absence of one of the engines, it is mandatory that this alignment exists. This constraint is required by the process of *Select Strategy*, which occurs in run time. A strategy to be applied to the current mood at any instant must exist.

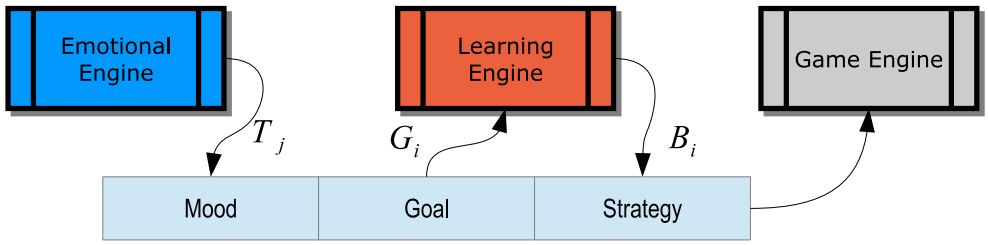


Figure 3.3: AGCBAR Architecture: Goal and Strategy Matching. Each of the goals, G_i, creates a strategy, B_i, which provides the goal, G_i, to achieve by the Learning Engine.

3.2.1 AGCBAR Architecture: Engines

As we described previously, the AGCBAR Architecture is composed by three engines: *Game Engine*, *Emotional Engine* and *Learning Engine*. These engines must meet certain features and requirements when they are implemented in a design. In the Chapters 4 and 5 we show one specific implementation for an Emotional Engine and a Learning Engine, respectively, and in the Chapter 6 we describe a video game which is implemented to present a complete integration of the AGCBAR Architecture with a Game Engine. Still,

the proposed architecture is intended to be applicable to many implementations of these components. In this section, we describe the functionality and the requirements that the implementations of these engines must fulfill.

Game Engine

The game engine is in charge of the simulation of the core rules of a particular game. It must create the environment of simulation and it has to populate this environment with the characters of the scenario. This is the basic component of all video games. The basic game loop is based on the perception-deliberation-reaction cycle, where each character in the scene perceive the current state of the environment, his reflection about it and his next action-decision-making. After the reception of the actions of all of the characters, the game engine simulates the changes in the environment, derived by the set of actions, throughout the physics and rules implemented in it.

Emotional Engine

The Emotional Engine is designed for the affect representation and synthesis according to the environmental perception of the characters. The main goal of this component is to enable a more realistic behavior of the character. Although there are cases in which the video games have no necessity of using any emotional or affect simulation, some other video game genres would greatly exploit the possibilities of this kind of engines. Thus, the affect simulation can enrich the behavior of the characters and situations that can appear at the environments created in this kind of video games (such as RolePlaying Games (RPGs) or "Sim Kind" games).

The Emotional Engine component has the objective of providing the Mood Dynamics that are necessary for the affect representation and simulation. The Emotional Engine component must receive the corresponding emotional events, E_i, with sufficient parameters for the identification and evaluation of the emotions that the character must experience, and produce a mood state, T_i.

The Emotional Engine, as part of the AGCBAR Architecture, must meet certain requirements, as in the case of the Game Engine, but also, it must fulfill the requirements of the Mood Dynamics. These requirements derive from the Cognitive Psychology [1], and from the Video Game development environment[2]:

[1] In the context of applied psychology, the appraisal theories usually establish some key points for the computational analysis and/or simulation of emotions, such as, the difference between the appraisal and the inference about of the events that a character perceived [46], minimal necessity of certain inferences to distinguish between emotions [81], believability and empathy with the user [21], and some others.

[2] The video game development is a very time-constrained environment. The development cycle, usually, is constrained by hard release deadlines and/or by some bottlenecks in the work flow of the production processes. Furthermore, at run-time, we have also some restrictions about the amount of time and resources that we can use for certain tasks, such as the artificial intelligence techniques. Thus, we have to make

- **AGCBAR-EmoR1**: The Emotional Engine must have a complete mapping of the moods, $T_i \in \mathbf{T}$. So there is no possibility that a mood state reachable by the characters emotional, not to be specified in the *Moods Dictionary*. The mood is the output of this engine, so the moods produced here are the ones that are going to be used in the Strategy Selection process.

- **AGCBAR-EmoR2**: The current mood state, T_i, of the character must influence in the Mood Dynamics. Thus, the Emotional Engine is intended as a function that takes the emotional events and the current mood to produce the new one[3]:

$$EE : \mathbf{T} \times \mathbf{E}^n \rightarrow \mathbf{T}$$
$$EE(T_i, \{E_i, \ldots\}) = T_{i+1}$$

- **AGCBAR-EmoR3**: As a consequence of the AGCBAR-EmoR1 and AGCBAR-EmoR2, the Emotional Engine must be initially configured with a starting mood state T_0. This mood must be the representation of the initial base tendencies of the character. These tendencies are described in psychology as the "personality".

- **AGCBAR-EmoR4**: Thus, the Emotional Engine must provide a model for defining the initial tendencies of the character facing his environment [50]. The character, in his profile, sets his own personality and, consequently, his initial and central mood towards which, in absence of emotions, the character tends to be.

- **AGCBAR-EmoR5**: So, there is a necessity to recover the initial mood state as part of the Mood Dynamics, this recovering represents the raising and lowering of an emotion, providing the attenuation of the past emotional events and the natural tendency to the neutral mood position [7].

- **AGCBAR-EmoR6**: The Emotional Engine must provide a mechanism to process the Emotional Events produced, which are described at the *Events Dictionary*. The way of treating these events is character-dependent, but all these events must be accepted by any character, thus we ensure the robustness of the architecture.

- **AGCBAR-EmoR7**: The computational cost for the transitions and evaluations of the moods and emotions must be moderated in order to be applied under soft real time constrains. The Emotional Engine is the responsible of *Mood Dynamics* concern, which is, by definition, a feature to be provided on-line while the game is running.

---

the Emotional Engine suitable for being applied as part of the development process and in the real-time execution.

Learning Engine

The Learning Engine, as opposed to the Emotional Engine, works during the design and development phase of the video game. The Learning Engine is in charge of the assisted creation of the behaviors so that they are adapted to the environment state and the current goal of the character. This engine must produce different behaviors extracted from the evaluation of the different strategies explored in a, possibly long, simulation process.

The Learning Engine can be implemented in many different ways but it is important to identify the fact that the primary objective of these implementations must be the creation of different strategies from different scenarios in which the environment changes due to the interaction of a set of agents that can cooperate or not.

As in the case of the Emotional Engine, the implementation of the different Learning Engines is constrained by the fulfillment of certain requirements. These requirements directly derive from the interfaces that the Learning Engine is connected to. Hence, the actions A_i selected by the strategy B_i must be of the set **A** described in the *Action Dictionary*, so the Game Engine can correctly interpret and implement them as actions in the simulated environments. The environment parameters, $\overrightarrow{S_i}$, that the Game Engine provides, are the only input of the environment state that the Learning Engine has. These parameters must be used to evaluate the current state according to the objective and to select the next action to be performed by the character.

3.3 AGCBAR Architecture. Dynamic View

The AGCBAR Architecture, as we presented before, is composed by different components that exchange information in order to produce the desired features of *Mood Dynamics* and *Automatic Controller Creation*.

The communication between the components is made by means of a set of interfaces, as presented in the Figure 3.4. They provide the information produced by a component and used by another. In order to support this communication, the interfaces must use the information described in the Architectural Dictionaries. These interfaces are:

- **Events** Interface: this interface represents the data exchange between the Game Engine and the Emotional Engine. The events, E_i, will be produced as instances of the *Event Classes* described in the Event Dictionary. They will provide the parametrized information taht is necessary for the processing of these "emotional events" in the Emotional Engine.

- **Mood** Interface: the current behavior, to be applied in the decision-making process, is selected according to the current mood, T_j, produced by the Emotional Engine. This interface is used by the Strategy Selection process for choosing the strategy to follow while the architecture is running the *Mood Dynamics*.

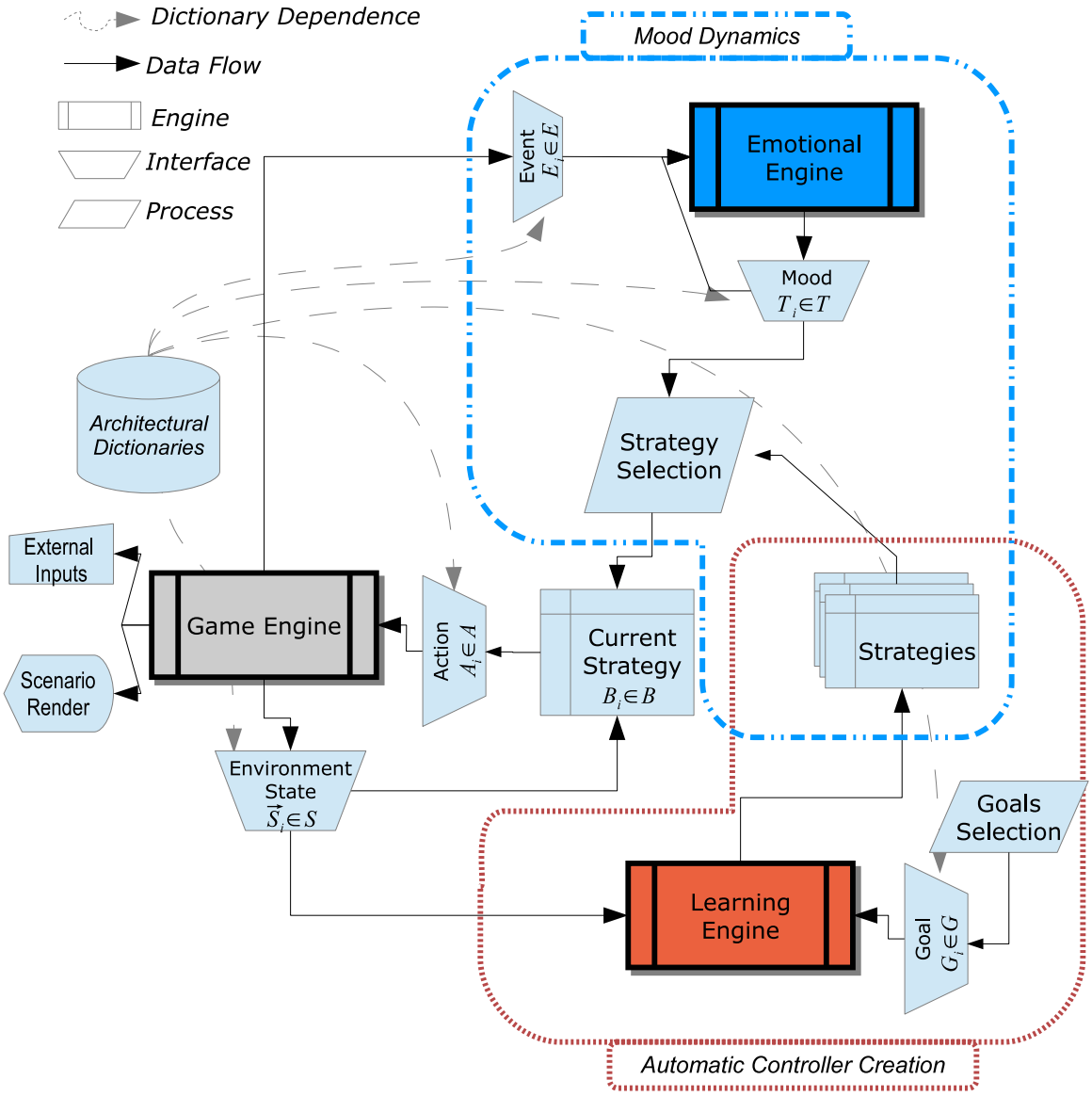


Figure 3.4: AGCBAR Architecture: Dynamic View. Displays the relationships among components through the interfaces.

- **Environment State** Interface: the simulated scenario and environment will change their state variables, $\vec{S_j}$, along the time driven by the characters' actions as well as by some internal events. This state is represented in a data structure shared between the Game Engine and the Learning Engine. Thereby, the *Automatic Controller Creation* feature can use it to produce the strategies suitable for each given scenario state. These state variables represent different parameters that reflect the state of the environment in a particular instant. The class and values of these variables are dependent of the game environment. This interface is thus the way to summarize and characterize the

environment state to make it usable by the Learning Engine and the Current Strategy to decide which is the next action to execute.

- **Action** Interface: describes the representation of the actions available for the controller to perform a particular strategy. The Actions, A_k, selected by the current strategy which we are used in a specific instant are sent to the Game Engine to perform the atomic actions described in the simulation engine which represent the core of the game. In some cases, the action produced by the Strategy is an abstract or parameterizable representation of an action, therefore, the Action Interface enables the communication between the Current Strategy, B_i, and the Game Engine.

- **Goal** Interface: this interface provides the mechanism to communicate the Goal Selection process with the Learning Engine, the current objective to achieve throughout the learning process. The goals reflected in the Goal Dictionary, are translated with this interface in the format that the Learning Engine needs, for instance, given some optimization learning engine, the Goal Interface can translate a particular goal, G_i, in a function to be maximized by the Learning Engine, $f_{G_i}(x)$.

3.3.1 Concerns Application Phases

As we said before, the two concerns addressed by the AGCBAR Architecture are complementary. They can be included in a development in order to semi-automatically create the controllers for the characters, which will react (or strategically act) driven by their current emotional state.

The AGCBAR Architecture features are based on the interaction of certain components in a given instant of time. In the Figure 3.5 we represent the different components that interact. This interaction is divided in two instants of the life cycle of the software. First, we have the development time. During this stage, the *Automatic Controller Creation* concern takes place. We address this stage as Off-Line Strategy Generation. Second, while the game is running we describe two processes or stages: Run-Time Action Selection and Run-Time Strategy Selection. In these two stages the strategies created during the development are used to act according to the mood state produced by the *Mood Dynamics* and the run-time simulation given by the Game Engine.

Off-Line Strategy Creation

This stage is part of the game development phase (design and implementation). Video game designers are producing the contents for a new game and require some support tools to automatically produce or assist in the crafting of these contents. During the development phase, the AGCBAR Architecture supports the creation of strategies with the *Automatic Controller Creation* concern. The Learning Engine implements this feature proving an automatic mechanism to produce character control routines, see Figure 3.6.

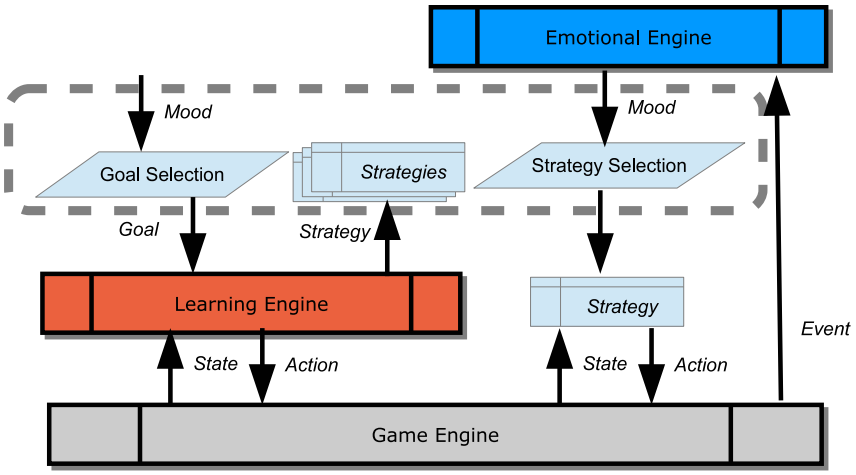


Figure 3.5: AGCBAR Architecture Components. During the run-time the Emotional Engine will produce the Mood Dynamics and at the development time the Automatic Controller Creation.

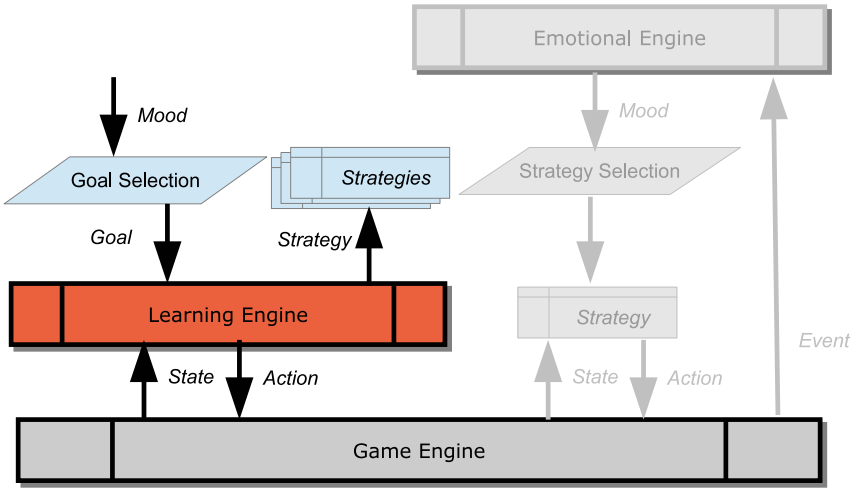


Figure 3.6: AGCBAR Architecture Components. The Learning Engine build the strategies according to the goal to achieve.

We define the Strategies, $B_i \in \mathbf{B}$, as the control routines used by the characters to perform the actions during the run-time simulation. These strategies are a probability distribution over the space of action, $\mathbf{A}$, given a state of the game, S_k.

$$B_i : \mathbf{A} \times \mathbf{S} \longrightarrow [0, 1]$$
$$(A_j, \vec{S_k}) \longrightarrow P(A_j, \vec{S_k}) = \pi$$

Therefore, as shown in the Figure 3.7, during the strategy creation process, the Learning Engine must evaluate the state of the scenario provided by the Game Engine. Then, after classifying this state, the Learning Engine selects an action, A_i, as the next

one proposed by the character controller. The selection of the action is carried out by the Learning Engine which evaluates the current state and selects the most suitable action to accomplish the current Goal of the character, G_i.

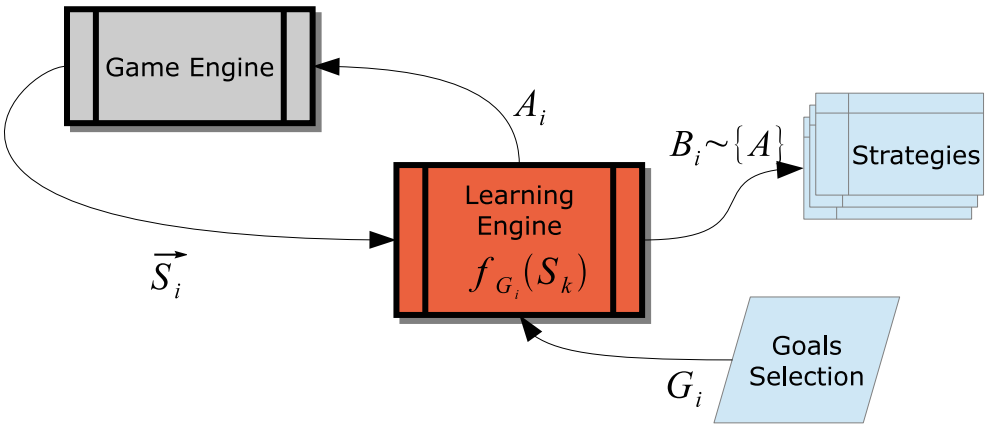


Figure 3.7: AGCBAR Automatic Controller Creation: Learning Engine. The learning algorithms need to evaluate the environment state $\vec{S_k}$ to adjust the strategy selecting the action A_j to achieve the current selected goal, G_i.

All the above mentioned procedures should be carried out in an iterative way for certain amount of episodes, during these episodes the Learning Engine adjusts the strategy while evaluating the current objective. When this learning process ends, the resultant strategy, B_i, is stored and associated with the goal, G_i, so it can be easily selected during the execution of the game by the *Strategy Selection* process, see Figure 3.8.

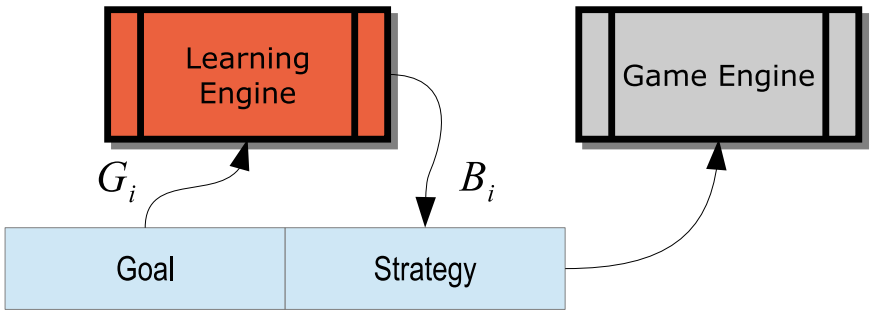


Figure 3.8: AGCBAR Automatic Controller Creation: Goal and Strategy Matching. Each of the goals, G_i, creates a strategy, B_i.

Run-Time Strategy Selection

When the game is running, the affect-driven behaviors are applied to enhance the sense of realism of the characters and their reactions. It is during this phase when the *Mood Dynamics* concern is accomplished, as shown in Figure 3.9.

The player expects that the characters involved in the scene that he is playing, show emotional reactions. The role of the Game Engine is to simulate the scenarios and according to the actions produced by the characters and some internal dynamics, to change the environment. These changes are sent to the Emotional Engine as *Events*, and they must prompt some emotional responses in the characters that perceive them. These events are evaluated and handled by the Emotional Engine, which will maintain the Mood of the characters along the time. When it is necessary (when it changes, when given a certain refresh rate or when explicitly asked), the Emotional Engine then returns the current *Mood* of the characters in order to use it for deciding the new *Strategy* to follow. The Strategy Selection process chooses the strategy according to the mood received.

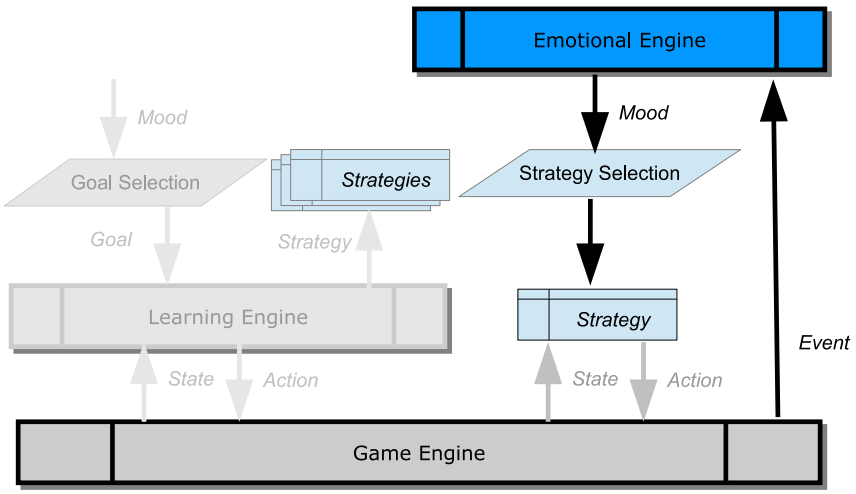


Figure 3.9: AGCBAR Run-Time Strategy Selection. The Emotional Engine evaluates the events received from the Game Engine to provide the Mood to Strategy Selection process

The *Mood Dynamics* concern produces the following sequence of actions in the different components that are included in this Run-Time Strategy Selection stage, see Figure 3.10

1. The action of a character, A_k, must be processed by the Game Engine in order for it to change the environment state, $\overrightarrow{S_j} \rightarrow \overrightarrow{S_k}$.

2. The Game Engine instantiates the events, E_k, related to the potential change in the environment state that are relevant for the mood analysis. They are communicated to the Emotional Engine.

3. All these events include the parameters that are relevant for the analysis of the emotions, such as source or target of the event, time-stamp, etc. The specific implementation of the game engine and/or Emotional Engine can force the inclusion of more parameters which must be specified during the design of the Emotional Event Interface, see Figure 3.11.

4. The Emotional Engine applies the events $\{E_k, \ldots\}$ to the current mood state T_j, producing a new mood, T_{j+1}.

5. The new mood is used by the Strategy Selection process to update the current strategy, according to the change of the mood.

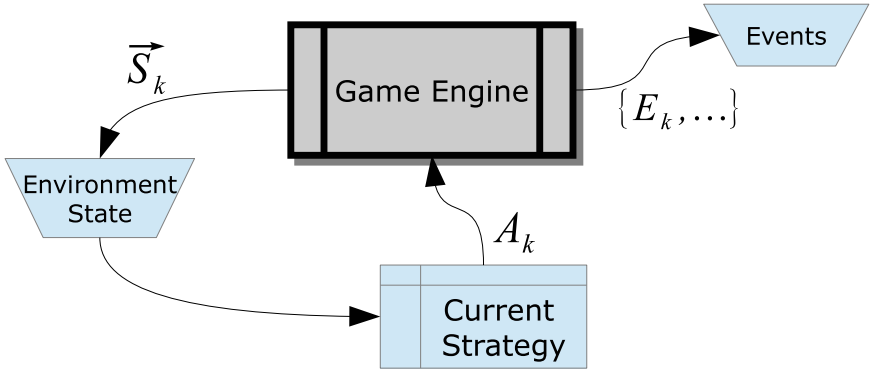


Figure 3.10: AGCBAR Mood Dynamics: Game Engine. The simulation produces the emotional events $\{E_k, \ldots\}$

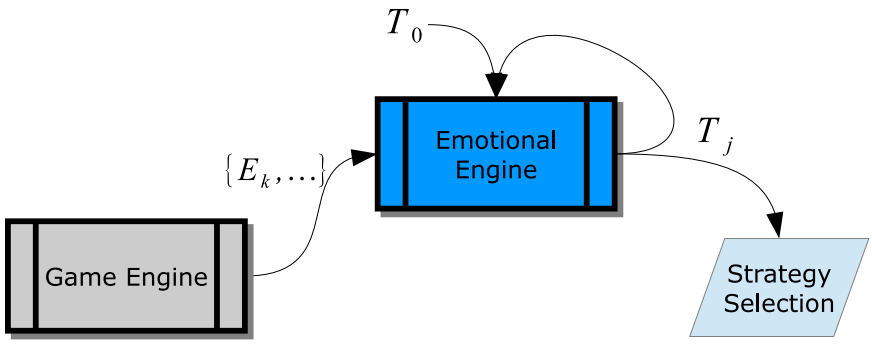


Figure 3.11: AGCBAR Mood Dynamics: Emotional Engine. Given an initial mood T_0, which describe the initial tendencies of the character. The Emotional Engine processes the emotional events, $\{E_k, \ldots\}$, to modify the current mood, T_j.

Run-Time Action Selection

Regardless of whether the *Mood Dynamics* feature is implemented or not, during the execution of the game, the current strategy that the character is using to manage his behavior is repeatedly queried about the next action, A_k, to perform. The strategy provided by the Strategy Selection process chooses the next action according to the probability distribution associated to the current state sent by the Game Engine, see Figure 3.12.

The basic loop of a video game, see Figure 3.13 can be described as:

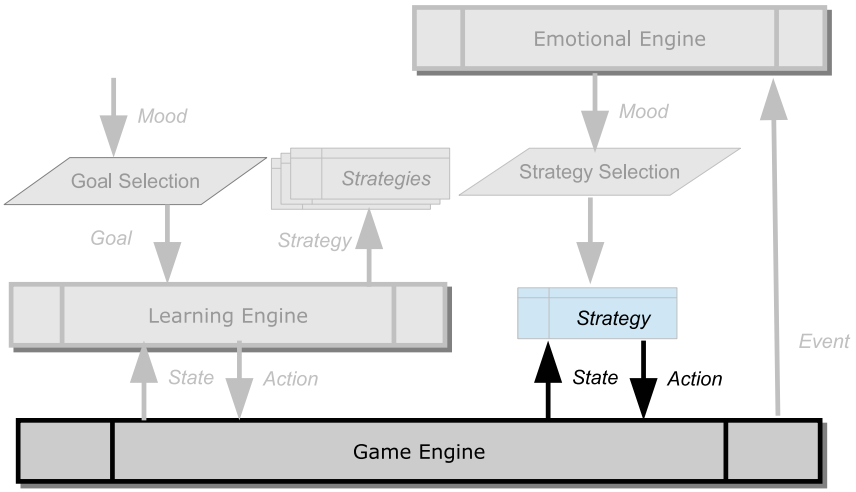


Figure 3.12: AGCBAR Architecture Components: Action Selection. The Game Engine asks for the next action to perform to the current Strategy.

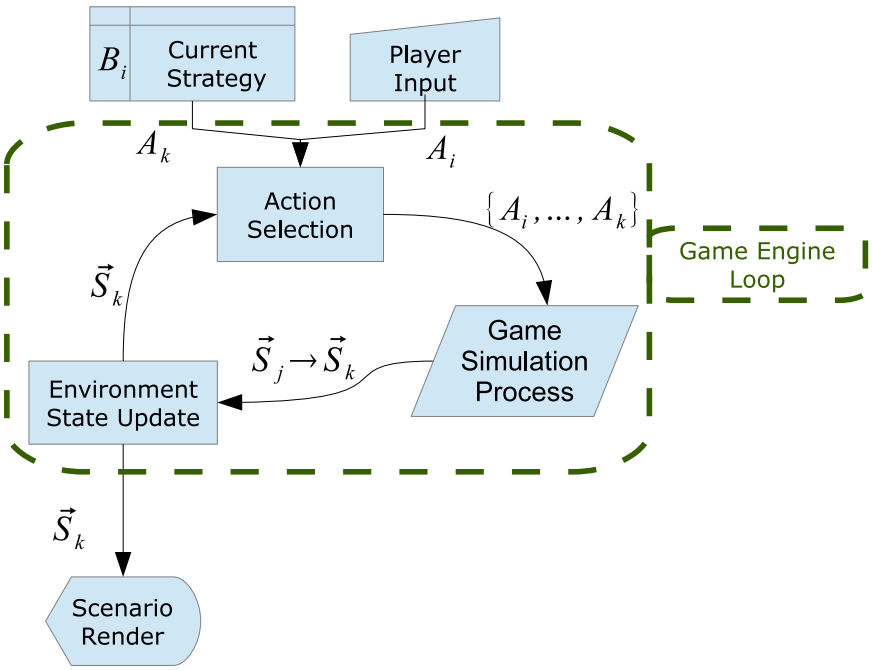


Figure 3.13: AGCBAR Basic Game Loop. The general game loop is based on the action-selection/simulation/state-feedback cycle. The character controllers must interpret the environment state in order to select their next action, A_k, to do.

1. The game scenario in a particular state is described by a vector of environmental variables, $\overrightarrow{S_j} = S^1, S^2, \ldots, S^n$.

2. This state is communicated to the characters in the scene. These characters decide which the next action will be, according to their current strategy, B_i (this is the

strategy driving the actions of the character i).

3. The game engine receives a set of actions of all of the characters in the scenario, $\{A_i, ..., A_k\}$,
4. These actions make the environment transit from one state to another, $\overrightarrow{S_j} \longrightarrow \overrightarrow{S_k}$. This is done by following the rules that describe the game mechanics.
5. This new state is again communicated to the characters.

In parallel, the Game Engine must send the scenario to the render of the scene to display it to the human player, and also, it must included the actions selected by the player in the simulation loop.

3.4 AGCBAR Architecture Design and Implementation Guidelines

The AGCBAR Architecture, as we described in the previous section, is a general architecture that can be applied to a wide range of video game developments. In this section, we propose a set of design and implementation guidelines which can help to deploy the components of the architecture in a video game development.

First of all, the game designer must decide if he needs to include both concerns, *Mood Dynamics* and *Automatic Controller Creation*, into his game. The AGCBAR Architecture concerns are designed to be independently applied. Hence, these features can be included into any development, but in some cases, the video game that is going to be developed may not need one of them.

The selection of the features to be applied leads us to four possible implementation scenarios for the AGCBAR Architecture, as shown in Figure 3.14. If the designer decides to use the *Mood Dynamic* feature, the implementation of the Emotional Engine and all of its features is mandatory, because the AGCBAR Architecture will need it when the game is running. On the other hand, the selection of the *Automatic Controller Creation* requires the implementation of the Learning Engine. Thus, off-line training of the strategies can be accomplished. In the case that the design of the video game includes both concerns, the complete architecture needs to be implemented. Therefore, both phases of the AGCBAR Architecture, Run-Time Strategy Selection and Off-Line Strategy Generation phases, will be applied. In any case, the phase of the Run-Time Action Selection is always present because it is imposed by the Game Engine loop.

Once the designer selects the features to be applied in the new development, a sequence of actions must be taken to implement them. As we show in the Figure 3.15,

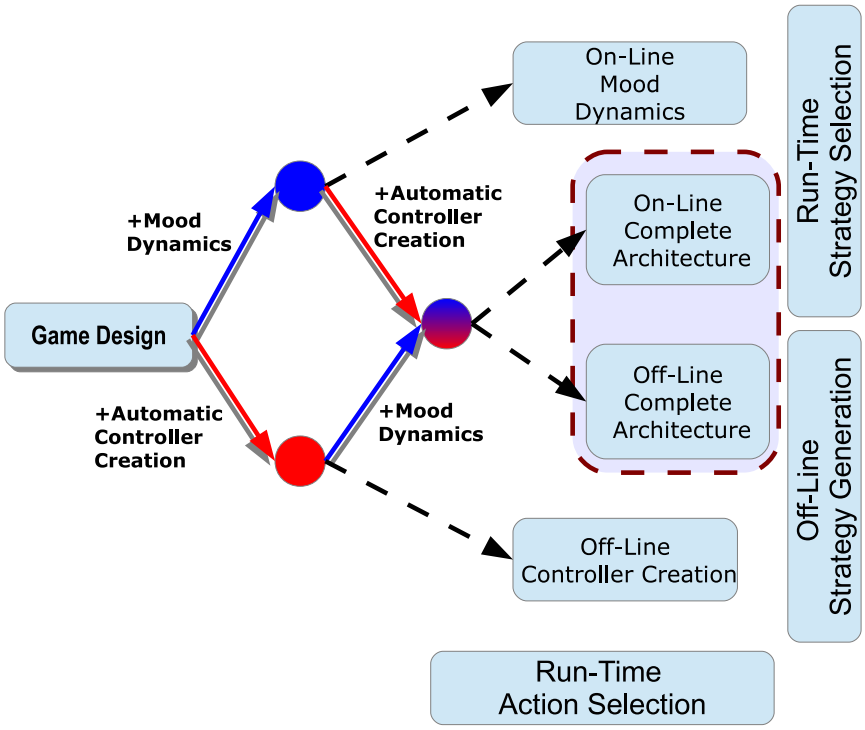


Figure 3.14: AGCBAR Architecture: Concern Composition. The presence of one or another of the concern produce the appearance of certain phases of application of these concerns.

each of the concerns has certain tasks to develop. The description of these tasks and their implementation guidelines are described later on the section. Basically, the *Mood Dynamics* has to describe the dictionaries and interfaces related to the Events Classes and Mood Tags, and the process of selection of the strategies to apply according to the current mood. The *Automatic Controller Creation*, for its part, is related with the dictionaries of Environment State, Actions and Goals (see the Figure 3.2 where the dictionaries and engines are associated). Also, the development of the Learning Engine needs of the Environment State interface and the Goal Selection process to work.

The **first step**, previous to the implementation, is **the definition of the *Architectural Dictionaries*** (see the Section 3.2 for their description). After the specification of the dictionaries, the interfaces of the architecture must be implemented according to the features selected (see Figure 3.15). As we said in the previous section, there are five main interfaces in the architecture: Emotional Event Interface, Current Mood Interface, Environment State Interface, Action Interface and Goal Interface. All of them must be implemented to support the communication among the components included in the architecture design.

Finally, the implementation of the engines referred to by the AGCBAR Architecture is closely related with the algorithms that we want to apply to resolve each of the concerns of the architecture. In this section we show the elements of the architecture that

must compose each implementation setup. This is also the interfaces and the requirements necessary for the integration of the whole system. In the following chapters (Chapters 4 and 5), we present two specific implementations for the Emotional Engine and for the Learning Engine, respectively, and in the Chapter 6 we present a complete Game Engine implementation which integrates all the components of the architecture.

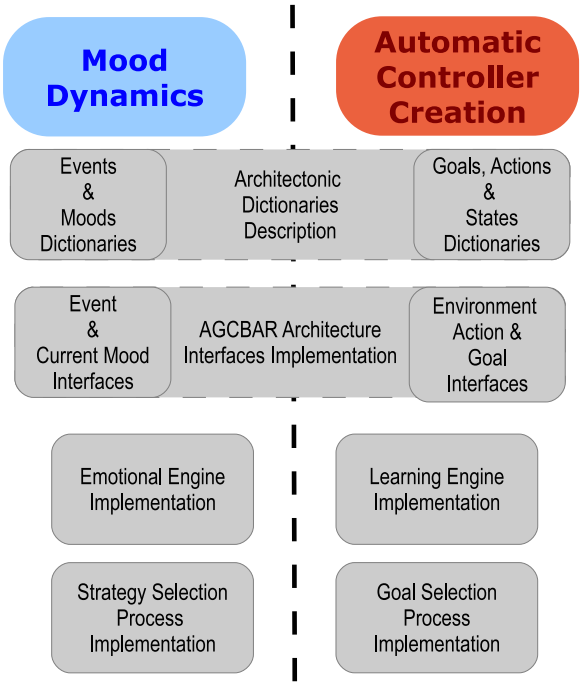
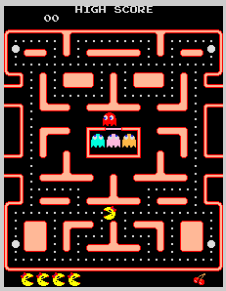


Figure 3.15: AGCBAR Architecture: Concern Implementation. The implementation of each of the Concern has associated a set of elements to implement and configure.

In order to illustrate the implementation of a video game using the AGCBAR Architecture, we will describe a simple video game, based on a well–known game. The game that we propose is a "Emotional-Ghost Ms Pac-Man™". The main idea of the game is a 2D maze, as in the figure (Screenshot of Ms PacMan, a clone of PacMan). In this maze there are two teams: one team of 4 Ghosts, which is controlled by the computer, and another formed by MsPacMan, which is controlled by the player.



The game is divided in levels, each level having its own different maze. The mazes are formed by walls that form corridors, one ghost's spawning cell and two or more crossing tunnels. The corridors are filled with pills. Each pill can be eaten by MsPacMan, scoring a certain amount of points. When all the pills are eaten by MsPacMan, the current level is clear, and the player proceeds to the next level.
The goal of MsPacMan is to make as many points as possible, clearing all the pills that appear in the maze. The goal of the ghost is to try to chase MsPacMan and to eat her as many times as possible.
There are 4 special pills on each maze, called PowerPills. If these pills are eaten, the Ghosts became "edibles" (i.e. they can be eaten by MsPacMan) for a fixed amount of time, or until the Ghosts are eaten. When a Ghost is eaten, it returns to the spawning cell and his chase starts again.
In this Chapter we use this game to show the description of the elements that compose the AGCBAR Architecture and how to describe them in this simple game. It is clear that in such a basic game there is no necessity for the proposed architecture, but we use it because is a well known and easy to understand game.
Our main objective is to apply the AGCBAR Architecture to the automatic construction of the Ghosts' controllers. Applying the emotional simulation to them to recreate more "emotionally driven" Ghost which feels fear or anger due the events of the game.

3.4.1 Events and Moods Dictionaries

The common implementation of the dictionaries is the textual enumeration of tags that have semantic meaning for the Events and Moods. The Events will be related to actions, consequences and/or objects that appear in the scene. They are described as classes formed by a key identifier attribute. The key point for their description is to identify the elements that can be relevant in the emotional state of the characters that we have in the environment. For instance, the death of some of the characters or the achievement of a certain task can be interesting for the emotional state of the characters, but the movement of some secondary agent (like a bird or rabbit) in the scene may not be relevant (in any case is the game itself the one that make something relevant for the emotional state simulation or not). But also, as can be inferred from the examples, the event classes have also some other attributes that are relevant for the parameterization of the events, such as the identifier of the dead character, or the specific location of the achieved task.

This definition of the Event Classes induces the appearance of the *Event Hierarchy* which will be included and extended in the implementation of the Emotional Engine in order to ease the integration of this engine in the architecture.

The Moods of the characters are utterly derived from psychology terms, and the

representation of the moods and their meaning is closely related with the Emotional Engine implementation. Because of that, we can not describe moods in the dictionary that are not going to be addressed by the engine. Therefore, the Emotional Engine provides a representation of the moods, and we must select a set of the ones that are complete over the possible space of moods, so there can not be a mood returned by the Emotional Engine that we do not have represented in the Moods Dictionary.

For the game that we presented before, we describe the following elements for the Events and Moods Dictionaries:

Events Dictionary -E-	**Moods Dictionary -M-**
Eaten(GhostID)	Angry
PowerPillTaken(PillID)	Neutral
Move(GhostID,Pos)	Afraid
Spawned(GhostID)	

All of these moods need to be supported by the implementation of the Emotional Engine.

■

3.4.2 Environment State Dictionary

The environment state is the set of variables that represent the actual scenario configuration, the positions, states and values of the elements that conform the simulated environment in a specific instant. In order to represent this configuration, a set of environment variables must be described. These variables summarize the state of the environment, so they can provide enough information to the Learning Engine when the learning process is going on.

The dictionary has the information about the variables to represent the environment, but it also provides a descriptive information about certain states that are relevant for the characters' controllers that we want to create. These descriptive states can be used by the Learning Engine as a discrete representation of the environment state so it can be easily used by the learning algorithm enclosed in the engine. This discretization of the environment is not mandatory but it can be helpful when the environments are complex.

In our proposed scenario, the game state is represented by a complete scenario enumeration. We have a scenario composed, at each instant of time, by a map where we have a set of cells. In each of the cells, we have an enumeration of mutually exclusive values which represent the different states of the cell.
These states can be:

Possible states of a cell in the map	
Empty	Wall
Pill	PowerPill
$Ghost_{1,2,3,4}$	$MsPacMan$
$EdibleGhost_{1,2,3,4}$	Spawn Cell

Additionally to this game board information, we include in the state representation:

- the current edible time,
- the scores of MsPacMan, and
- MsPacMan's remaining lives.

This describes the Environment States as a vector of parameters
$\vec{S_i} = \langle S_i^1, S_i^2, S_i^3, S_i^4 \rangle$

Where,

S^1	A matrix of $n \times m$ of enumerated values.
S^2	Edible time in seconds as an integer in $[0, 10]$
S^3	The score of the player as an integer in $[0, 100000]$
S^4	The number of lives remaining $[0, 3]$

3.4.3 Emotional Event Interface

Emotional Events are sent by the Game Engine according to the changes in the environment. These events must be treated by the Emotional Engine, so the implementation of this interface must be guided by the needs that we have from the Emotional Engine. The Event Classes, E_i, are instantiated with the corresponding parameters through this interface. The Event Classes can be extended to be adapted to a particular Emotional Engine implementation, in order to provide all the parameters needed by the Emotional Engine. These instances are sent to the Emotional Engine which process them.

In our example game, the events are treated as a string with the name of the event specified in the Event Dictionary. These strings will be sent to the Emotional Engine to process emotional events. Additionally, we include the source and target (if they exist) of the event.

Eaten: Src ⇒ *Ghost1*: Tgt ⇒ *MsPacMan*
PowerPillTaken: Src ⇒ *MsPacMan*
Moved: Src ⇒ *MsPacMan*

■

3.4.4 Mood Interface

Once the Emotional Engine processes the emotional events, the current mood of the character may transit to another. Even if the mood does not change, the Emotional Engine communicates this Mood of the characters in order to select the strategy to apply in the decision–making process of the character. The current mood is provided as a tag that describes the emotional state of the character. These tags, as in the case of the emotional events, must cover all the moods described in the *Mood Dictionary*. It is also possible that the mood state is represented some other way depending on the implementation of the Emotional Engine, because it is used as part of the emotional evaluation of the events.

The Strategy Selection process receives the current mood to decide the strategy to apply afterwards, the update of the mood state can be done by request, when we need to update our strategy, or by event, when the mood change is notified to the Strategy Selection process component, which will change the strategy accordingly.

In our case, the representation of the moods will be done as in the case of the emotional events, with a tag, $T_i \in T$. For instance:

Angry **Afraid**

■

3.4.5 Environment State Interface

The Environment State Interface is the representation of the Environment State variables that is communicated from the Game Engine to the Learning Engine during the *Automatic Controller Creation* and to the Strategy during the game execution to decide the actions to be taken in a specific behavior.

This representation generated from the state variables as Environment State Interface must have a balance between information and specificity. Due to this trade-off, the variables are usually summarized or discretized in ranges, or a mechanism of representation of the information is used to make the learning process easier. The learning process of the Learning Engine is closely related to the dimension and complexity of the state representation and the action set. Thus, this interface is critical in the design of the *Automatic Controller Creation* facet.

The environment variables described in the Environment State Dictionary are summarized as follows:

- the scenario is described as it is, with the matrix representation of the map with an enumerated value on each cell according to the elements that has in.
- the ghost that are edible.
- the remaining number of lives of MsPacMan.
- the Edible Time is set in 3 slices: *Last2sec*, *HalfTime* and *FullTime*.

3.4.6 Goal Dictionary, Interface and Selection Process

As we said, in the Off-Line Strategy Creation Phase, the learning algorithm implemented as Learning Engine will create a strategy combining different actions according to the environment state. Each strategy is created to achieve a particular Goal G_i, which is also associated to a specific Mood T_j. The most basic mechanism for selecting the goal to achieve, taking into account that we must create a strategy for every goal, is to take each mood state, to select the goal assigned to it, and to learn a strategy for this goal. Then we can take another mood and so on.

Therefore, the Goal Interface provides the mechanism to adapt the Goals, $G_i \in \mathbf{G}$, described in the Goal Dictionary to the format or representation needed by the implementation of the Learning Engine used. For instance, the Goal Interface can associate the goal, G_i, an utility function, f_{G_i}, to evaluate performance of the strategies that are being created.

Learning Plans

It is difficult (with a reasonable computational power) to learn a whole strategy without guidance. The application of the machine learning algorithm to a very complex environments with a huge number of parameters, continuous actions, etc. is shown to be hard in terms of computation. The inclusion of the scenarios in the cycle of the *Automatic*

Controller Creation is oriented to help the planning of different environments in which we want to explore a specific strategy given certain Goals represented by the environment.

For instance, we can create a scenario in order to learn how to escape from a dangerous encounter. We set the environment up so it has a escape zone, some particular enemies or threats and a specific set of actions. We maximize the reward when we escape, encouraging the learning of the special strategies to resolve these situations. Then we save these strategies to a library of different plans and strategies to be used in different situations.

3.4.7 Action Dictionary and Action Interface

The strategies, B_i, used by the characters under the AGCBAR Architecture, are composed by the actions, A_i, described in the Action Dictionary. The Action Interface has the responsibility to adapt this action space A into the action space supported by the Game Engine. These two action spaces may be of different granularity or level of abstraction. Indeed, strategy definition should be more abstract in order to deal with compound actions, while the engine requires a set of actions closer to the actual engine semantics.

Hierarchy of Actions

The number of possible actions in a video game environment could be big. Usually, the exploration of the different strategies becomes the exploration of the different actions that could be used in different states of the environment. The larger the set of actions the longer the time to explore them. In many cases, it could be helpful to implement a hierarchy of actions, that can abstract some details of the strategies.

Furthermore, these high level actions can be shown as a particular training scenario in which we want to learn how to best use the low level actions that compose the higher action. For instance, we may also have a high level action called "Find Escape Path" and then we have the different low level actions that are enclosed in it which represent where to go on the next step, so we can tailor a new training scenario only to learn which is the best way to "Find Escape Paths".

The actions A_i described in the Action Dictionary for the AGCBAR Architecture in the case of MsPacMan are translated into the actions supported by the Game Engine.

Continue....

■

...
The actions supported by the Game Engine are:

- Move Up,
- Move Down,
- Move Left and,
- Move Right.

Action Dictionary -A-	**Game Engine Actions**
Move Away	Direction opposite to MsPacMan
Move Toward	Direction to MsPacMan
...	...

These actions prompted in the Game Engine are based on the absolute position of the Ghosts and MsPacMan, but we choose to represent the actions A_i in a relative position way.

3.4.8 Strategy Selection Process

During the Run-Time Strategy Selection Phase, while the Emotional Engine is announcing the changes on the current mood state of each character, the selection of the strategy to be used by the character must be done. The strategy selection is done according to the Mood-Goal-Strategy mapping. As we said, for each mood, T_i, we must have a strategy, B_i, that tries to achieve the goal, G_i, related to this mood. For instance, if we have a mood called `Broken`, which represents a character that is so scared and worried that he only tries to flee and survive, we must have a strategy that performs the actions to try to achieve these abstract goal, for example running as fast as possible to the nearest exit.

As in the rest of the architecture design, there are different mechanisms of implementing this element. A simple mechanism to implement this process is to select directly the strategy related to the mood. This mechanism leads the character's behavior to a completely emotionally-driven behavior, where the strategy is determined by the current mood state. Thus, the decisions of the character only obey to his current mood goals.

In our example, we decide to implement the basic processes that assign to each Mood, T_i, a Goal, G_i and to associate them with a utility function, $f_{G_i}(x)$, through the Goal Interface. Hence, for each of the moods described on the Mood Dictionary we set the Goals and the Functions.

Mood	Goal	Goal Interface Function
`Normal`	Roaming Chase	$= f(distancetoMsPacMan) + g(distancetopowerpill)$
`Angry`	Frenetic Chase.	$= min(distancetoMsPacMan)$
`Afraid`	Maximize MsPacMan Distance.	$= max(distancetoMsPacMan)$

3.4.9 Game Engine

The Game Engine must be implemented according to the features that we want to include in the game development. The existence of these features provides the inclusion of the other engines of the architecture, so, the Game Engine must generate the necessary information to fulfill the interfaces of communication with the respective engines.

Therefore, the Emotional Engine needs the emotional events to produce the *Mood Dynamics*, in that case, the Game Engine must provide the events derived from the environments state transitions, sending these events to the Emotional Engine. Thus, the emotional events produced by the Game Engine:

1. Must have the information to enable the process of the emotions that are prompted. To implement this feature in the Game Engine, all the actions performed in the scenario must be classified in order to locate those actions that can produce an emotional event. It is important to remark that not all the actions and events produced in the Game Engine are emotional events.

2. While the game is running, at each time that the state changes, a set of emotional events must be sent to the Emotional Engine, the refresh rate of these events can be variable, but it is recommended that each of the emotional events has attached a time-stamp. This time-stamp will be used (probably, but depending of the Emotional Engine implementation) during the Mood Dynamics to provide the attenuation and flow of the emotions along the time (see the requirement **AGCBAR-EmoR5** of the Emotional Engine in Section 3.2.1).

On the other hand, when the *Automatic Controller Creation* feature is included in the design of the video game, the Game Engine implementation must provide two things:

1. The learning mechanism of the strategies for the creation of the controllers will require (depending of the implementation of the Learning Engine) the massive simulations of scenarios to perform this process. Thus, the Game Engine must have a mechanism to produce these simulations, probably in a batch mode, unattended and without rendering or input components.

2. The actions that we declare in the Actions Dictionary must be recognized by the Game Engine. This is done by the Action Interface. These actions can be complex or compound declarations of atomic actions. In this case, the engine must execute the associated actions in the environment that compose these complex actions. Hence, an action that describes a set of steps is reasonable for the construction of the strategies but must be implemented by the Game Engine with the corresponding atomic actions.

3.4.10 Emotional Engine

We present a detailed implementation of an Emotional Engine in Chapter 4. The implementation of this engine must meet all the requirements proposed in the Section 3.2.1. These requirements can be extended depending on the particular emotional model included in the engine. The requirements presented in this chapter are related with the mood state which is the critical element for the strategy selection process.

Therefore, the implementation of this engine must include the mechanism to provide the differentiated profiles for the different characters. Hence, the requirement **AGCBAR-EmoR4** demands that the Emotional Engine provides a mechanism for defining the initial tendencies of the characters, the implementation of these engines must enable the easy definition of these tendencies of the characters. Moreover, the Emotional Engine has to process the emotional events for all of the "sensitive" characters (those characters that perceive the events and will have emotional responses to them), and all of these characters have a current mood to use in the Strategy Selection Process.

3.4.11 Learning Engine

The Learning Engine implementation is closely related with the learning algorithm that we want to use for the strategy creation. There are a lot of techniques suitable for this purpose: neural networks, reinforcement learning, evolutionary algorithms, etc. In the Chapter 5, we propose an implementation based on a hybrid algorithm which includes soft computing and reinforcement learning.

In the implementation of the Learning Engine, it is important to remark the relevance of the action and state spaces. Bigger spaces (many actions or possible environment states) make more complex the strategy creation. Therefore, it is key to maintain a certain balance between these sets. The representation of the environment can be summarized in the Learning Engine to reduce the size of these state spaces, but it may incur in a losing of information.

Moreover, the *Automatic Controller Creation* feature is intended to produce a set of strategies aligned with the mood through the goals described. When the Learning Engine is implemented, the algorithm must have a mechanism to evaluate the current strategy (the one we are adjusting at that particular time) with the function that defines the current mood. This mechanism must guide the steps of the learning process. There are algorithms that try to imitate an existing behavior that is representative of certain goals. Other algorithms may only need to describe the desirable states and try to achieve those states through the actions. In any case, it is important to have in mind that this feature is designed to support the creation of complete strategies, but it is also possible to use the strategies created as part of more complex controllers that include other techniques to decide the actual behavior and action of the characters[3]. Even in this case, the strategies must be aligned with the moods if we desire the *Mood Dynamics* feature.

3.5 Discussion

According to Garlan [28] and the Software Engineering Institute[4], "*a Software Architecture consists of a component structure, the relations between components, and the rules governing the design and evolution of software systems.*"

Moreover, current software engineering trends point towards the use of design patterns, COTS components (Components Off-The-Shelf), product line engineering, reference and aspect-oriented architectures to further reduce production costs, to prevent project risks and to help build products with better quality [55].

In this Chapter we present the AGCBAR Architecture, this architecture addresses two different concerns. These concerns are not supported, in this moment, by any other architecture. According to Garlan we provide a description of the structure, relationships and rules used to solve our concerns. To solve them, we involve three different components that are included in the development and execution of a video game. By following the trend in the video game development, the architecture is designed to use any implementation, that meets the requirements, in each of the components. Thus, we can reuse off-the-self solutions to each of the components.

The *Mood Dynamics* concern can be addressed by some other emotional agent designs, but we have the objective of using this architecture in the video game development. This kind of development requires for the game designer to have a certain control of the behavior of the characters that are in a scene. Thus, the AGCBAR Architecture keeps the deliberation process of the action to be taken out of the emotional model. Hence, some emotional models, such as WASABI [7], present the embodiment of the affective agent in a BDI architecture. These models assume that the emotional agent has the deliberative

[3]We describe some of these possibilities in the Discussion (Section 3.5) of this chapter

[4]http://www.sei.cmu.edu/architecture

ability to decide which action to take according to his emotions. In our approach, we use a set of possible strategies (created by the game designer or, at least, supervised by him in development time) to perform the actions, and we use the mood state (i.e. the aggregated state of different emotions) as a parameter in the strategy selection process. So, the actions are described in the strategies created and do not come from emotional reactive responses that can be harder to handle by the designer. The lack of emotional reactive responses prevent the production of actions derived from primary emotions[5].

The events produced in the Game Engine, used by the Emotional Engine, are described as information exchanged by the different engines. Thus, the *FearNot!* model [21] needs the type, valence, target and cause (among others) parameters in order to be able to analyze the events. So, in the AGCBAR Architecture, the events produced by the Game Engine are produced with certain information that can be completed, adapted or extended by the Event Interface, so they can be used by the Emotional Engine that we implement in our development. Therefore, if we want to include in our emotional model "compound–attribution emotions"[6] we must produce the required information in the Event Interface to adapt the events to the description needed by the emotional model.

The *Automatic Controller Creation* concern leaves the learning process open to create strategies for different levels of abstraction. The AGCBAR Architecture provides a mechanism that allows us to include the creation of emergent behaviors for the video game characters. But these strategies are created off-line, in development time, so they can be tested and supervised by the game designer, in order to adapt them to the desired final behavior.

Additionally, alternative mechanisms for the strategy creation can be used aside of the learning mechanisms. These mechanisms can be included in the architecture providing different strategies like the *Learning Engine* does. This implementation of the AGCBAR Architecture substitutes the Learning Engine by a different mechanism of strategy construction. The strategies created by these other mechanisms must also be attached to the mood states described in the dictionaries, so we can ensure the Strategy Selection Process. So, we must have a strategy for each of the possible moods of the character, no matter how these strategies were created.

For instance, we can create a set of strategies, described by a Finite State Machine or Rule-based System. Each of these strategies is to be associated to a certain Mood as in the case of the learned strategies but discarding the *Goals* of the mapping (see Figure 3.3 in the Section 3.2).

Likewise, it is possible to include different implementations of the *Strategy Selection Process*, to include some baseline strategy, modified by a combination of strategies derived from the mood state. For example, if we take different moods and a basic strategy, we can

[5]Primary emotions as described in the Becker-Asano's WASABI model.

[6]In the OCC model: "the emotions produced after reasoning about the consequences and actions perceived".

use a strategy done by a weighted combination of the other strategies.

Chapter 4

Emotional Elicitation Process (*EEP*). A Model for Synthetic Emotions

> There are good and bad times, but our mood changes more often than our fortune.
>
> Thomas Carlyle

In this chapter we present the *EEP Model*. This model is designed as a component that can be used as the Emotional Engine in the AGCBAR Architecture, as described in the Section 3.2.1 in the previous chapter. Besides that, this model can also be used as a general emotional engine that can be applied to different architectures of video games and simulators.

In order to fulfill the requirements of the AGCBAR Architecture, the EEP Model is designed as a set of internal components that makes possible the interaction with the rest of the components of the architecture, implementing the interfaces specified by it. Although, the internal design of the model is made with a top-down approach to meet the requirements of the architecture, we take the bottom-up features that arise from the detailed study of the general emotional model designs.

This chapter is organized as follows: First, in Section 4.1, we describe the EEP Model as part of the AGCBAR Architecture, we also explain in detail how it fits in the abstract architecture. In Section 4.2, we describe the usage of the psychology concepts, introduced in Chapter 2, and developed by the EEP Model. In Section 4.2.4, we present the specific requirements for the EEP Model. In Section 4.3, we show the EEP Model internal architecture, we also describe the components that are part of the model. In Sections 4.4 and 4.5, we present two stand-alone validation tests of the EEP Model in video

game scenarios. Finally, in Section 4.6, we discuss the EEP Model, and we compare it with some other emotional models.

4.1 EEP Model as Emotional Engine in AGCBAR Architecture

The EEP Model is an instance of the AGCBAR Architecture Emotional Engine. As we describe in the Section 3.2.1, this is the component that implements the *Mood Dynamics* concern specified by the AGCBAR Architecture. This feature must produce the adequate emotional simulation from the events perceived by the characters in a video game scenario, so they can alter their internal "mid-term" Mood State. The alteration of their mood state will produce a variation in the goals the characters pursuit, and as a logical consequence, they also make en effective change in their strategy and behavior.

Therefore, taking the Figure 3.11 of the AGCBAR Architecture design, we decompose the Emotional Engine component in a set of elements that must be present in the internal design of this engine. We describe, as stated in the **AGCBAR-EmoR2** requirement, the Emotional Engine as a function $EE(T_i, \{E_i, \ldots\}) = T_{i+1}$, that must produce an updated mood (Mood Tag), T_{i+1}, after the reception of a set of emotional events $\{E_i, \ldots\}$ given a previous mood T_i. We establish the following steps to accomplish this transformation process:

1. *Emotional Event processing*: the events produced in the Game Engine, and described in the Event Dictionary of the AGCBAR Architecture, are received by the Emotional Engine. They must be evaluated in order to produce the affective response in the character. To do so, we must:
   - *Classify the events* according to the aspects that each emotional event represents. For that, each even must be decomposed in certain Emotions that the emotional engine can process.
   - *Quantify the events* because those emotional events produced in the environment and sent to the Emotional Engine have different intensities in terms of the emotion they represent.
2. *Mood Dynamics*: given the current Mood of the character, T_i, the Emotional Engine processes the emotions produced by the emotional events and changes the character's mood to a new one, T_{i+1}, accordingly.

By following these steps, we can produce the transition between different mood states. But in order to Emotional Engine to do so, the EEP Model needs certain components, see Figure 4.1, that will adapt the engine to the different characters and to the architecture. Thus, we must extend and/or adapt the Architectural Dictionaries to enable

a coherent integration of the EEP Model in the architecture. Moreover, the AGCBAR Architecture interfaces (Event and Mood Interfaces) must be implemented to keep a tight coupling between the inputs and outputs of the Emotional Engine as well as with the rest of the components of the AGCBAR Architecture.

If we want to adapt the EEP Engine to the different characters, as we said in the requirements **AGCBAR-EmoR4** and **AGCBAR-EmoR6**, the characters must have their own profile, which includes their initial tendencies, T_0, and their particular perception of the events, E_i, that occur in the environment. This requires the inclusion of a Character Profile and of the Conceptual Dictionaries (specific EEP Model dictionaries), because we need to represent the character's preferences in terms that are recognized by the EEP Model.

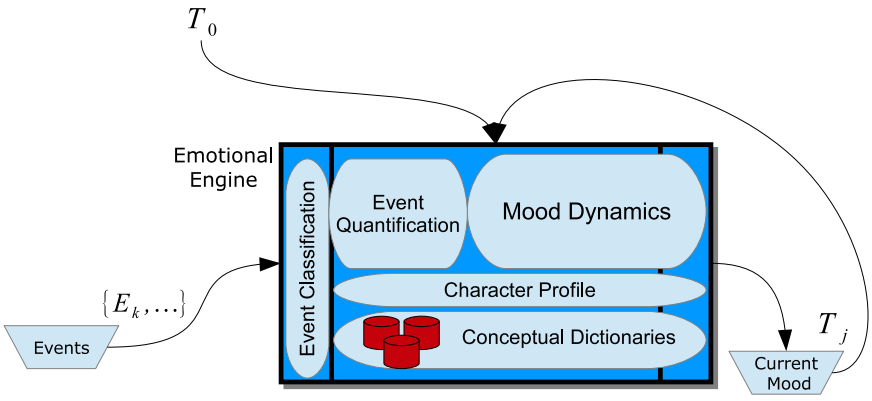


Figure 4.1: EEP Architecture Analysis. These components are necessary for the integration with the architecture and the processing of the emotions.

4.2 Usage of Cognitive Psychology Concepts in EEP

The design of the EEP Model is founded on top of some well-known psychology and cognitive theories, that were introduced in Chapter 2. In this section, we specify how important they are for the development of EEP Model, and how they are applied to the Event Classification and Quantification in order to produce the *Mood Dynamics* feature.

4.2.1 OCC Model

As in the case of the OCC model description made by Bartneck [5], we apply a process of analysis of events based on:

1. Classification: we analyze which emotions are prompted by a particular event.

2. Quantification: we evaluate the intensity of the emotions produced according to the standards (personality profile) of the character.

OCC Emotions	Group of Emotions	EEP Emotion Set
HappyFor Pity Resentment Gloating	`Fortune-of-others`	Consequence Emotions $\mathbb{C}$
Joy Distress	`Well-Being`	
Admiration Reproach Pride Shame	`Attribution`	Action Emotions $\mathbb{A}$
Gratification Gratitude Anger Remorse	`Attribution-Well-Being`	Compound Emotions $\mathbb{T}$
Hate Love	`Attraction`	Object Emotions $\mathbb{O}$
Hope Fear Disappointment FearsConfirmed Relief Satisfaction	`Prospect-Based`	**Not Used**

Table 4.1: Emotions used in the EEP Model. The selected set of emotions, $\mathbb{E} = \mathbb{A} \cup \mathbb{C} \cup \mathbb{T} \cup \mathbb{O}$,

3. Mapping and Expression: after the evaluation, the quantified emotions must be mapped into a representation that summarizes them.

The EEP model uses the OCC model to analyze the events, E_i, that a character perceives and to select a set of emotions prompted by each of these events. We use a simplified representation of the original model, according to the characteristics of the objective application domain. These characteristics are summarized in Table 4.1. Our approach seeks a trade-off between the provision of flexible mechanisms in order to define emotions, and the possibility to design and configure the model with a reasonable effort. Even more, the use of all of the parameters and emotional concepts described by the original OCC model is suitable to be incorporated to the EEP model. The mapping of the resulting emotions will be produced in a formal mathematical space of representation where the emotions and moods are integrated (see Section 4.3.3).

4.2.2 Big Five Personality Traits

The general psychological model of the Big Five Factors [48] (a.k.a. OCEAN) introduces different factors useful for the computational representation and manipulation

of character personalities. The five factors used by this model are:

- **O**penness: Appreciation for Art, Emotion, Adventure, Unusual Ideas, Curiosity, and Variety of Experience (inventive / curious vs. consistent / cautious)
- **C**onscientiousness: Tendency to show self-discipline, to act dutifully, and to aim for achievement; planned rather than spontaneous behavior (efficient / organized vs. easy-going / careless).
- **E**xtraversion: Energy, positive emotions, and the tendency to seek stimulation from the company of others (outgoing / energetic vs. shy / reserved).
- **A**greeableness: Tendency to be compassionate and cooperative rather than suspicious and antagonistic towards others (friendly / compassionate vs. cold / unkind).
- **N**euroticism: Tendency to experience unpleasant emotions easily, such as anger or anxiety, depression, or vulnerability. (sensitive / nervous vs. secure / confident).

The personality of a character sets the initial value of the character's state in a "neutral" mood, T_0. The personality does not changed easily and, for virtual characters, it may be considered constant. The EEP Model includes, in the profile of a character, a parameterized version of his personality (as the temperament representative information) described by the Big Five Factors representation[1]. This parameterization will be used as the starting point for the mood of the character.

4.2.3 *P*leasure-*A*rousal-*D*ominance Emotional & Temperament Model

The EEP model manages the PAD Model in different aspects: (1) the projection of the emotions of the OCC model into the PAD space (as shown at the Table 4.10). (2) To see the translation of the personality to the default mood (using the Mehrabian transformation [50], consult Table 4.4, which shows the reduced set of emotions used by EEP). and (3) the representation of the mood in the Mood Vector Space (see section 4.3.3).

4.2.4 Emotional Engine Requirements

Apart from the general AGCBAR Architecture requirements for the Emotional Engine described in Section 3.2.1, the implementation of the Emotional Engine component done in the EEP Model requires the fulfillment of additional requirements. These requirements arise from the different models and theories used to implement the EEP.

1. **EEP-Emo1**: The **AGCBAR-Emo6** requirement, which addresses the processing of the Emotional Events recorded at the *Events Dictionary*, makes the inclusion of a mechanism to analyze these events necessary. Lazarus' theory of cognitive psychology

[1]Described as real numbers between $[-1, 1]$

[39] describes an emotion as: "*psychological response produced after the thought produced by the perception of certain events*", so we need a mechanism that allows the Emotional Engine "think about events". The OCC Model provides this mechanism in the EEP Model, thus its structured analysis of the emotional events is used as baseline for the process of evaluation of the Emotional Events produced in the Game Engine environment.

2. **EEP-Emo2**: The use of moods, as described by the **AGCBAR-Emo1, AGCBAR-Emo2** and **AGCBAR-Emo3** makes it necessary to find an appropriate way to represent them. Also, because the emotions produced by the events modify the characters' current mood state, we need a mechanism to represent emotions and moods. We also need to find an algebraic structure to operate with them so we can transit $T_i \rightarrow T_{i+1}$. Mehrabian's PAD Models (Temperamental and Emotional) provide a homogeneous representation for both, emotions and moods. With these underlying models, we are able to create the Mood Vector Space, $\mathcal{M}$, (Section 4.3.3), so the Emotional Engine could handle those moods and emotions.

3. **EEP-Emo3**: The AGCBAR Architecture describes, through the **AGCBAR-Emo4** requirement, the necessity of providing an initial mood. This initial state will be used as the equilibrium point in absence of emotions. Thus, this base mood, T_0, will be considered as the long-lasting temperament[2]. In our model, this "temperament" is intended to be the character's personality and it will be handled using the PAD Temperament model and its translation from the Big Five Personality model[50]. We will explain later how the mood T_i translated to the PAD space will be projected to $\mu_i \in \mathcal{M}$.

At the end of this chapter, we present two application examples of the EEP Model in a video game.

- A combat scenario presenting the straightforward application of the selection of different strategies based on the mood state. (Section 4.4)
- A storytelling support application of the EEP Model creating dynamic storylines using the mood state of the characters as a guide. (Section 4.5)

Continue....

[2] "Temperament" is, thus, distinguished from mood, T_i, and it refers to the individual's stable or long-lasting emotional characteristics (i.e., emotional traits or emotional predispositions). More precisely, temperament is the mean of a person's emotional states across a representative variety of life situations. [51]

..

These two scenarios show the application of the EEP Model inside (in the combat scenario) and outside (in the storytelling) of the AGCBAR Architecture. For these examples we used the commercial video game NeverWinter Nights^{TM}(Bioware)[2].

> *Neverwinter Nights (NWN) is a computer game set in a huge medieval fantasy world of Dungeons and Dragons. This role-playing game (RPG) will place you in the middle of an epic tale of faith, war, and betrayal.*

Along this chapter we will use these scenarios to exemplify the implementations and designs of the components of the EEP Model. And at the end of the chapter, we will present the results and we will evaluate the believability and flexibility of the scenarios.

The first example we present is a ***combat scenario***. As in many other games, here too there are some series of confrontations among different sets of characters. These kind of scenarios are suitable for the appearance of emotions, prompted by stressful situations that will make some of the characters lose their wish to fight or will make them become enraged by the events of the battle. In our first example, we will present an encounter between a party of human adventurers and a defending band of orcs (in terms of the world represented by the NWN game we can say that the orcs are very belligerent and brutal).

The second example is a ***storytelling scenario***. In this scenario, we will apply the EEP model to the plot design of the story. We will use the NWN as framework only for the continuity purposes, but the real contribution of this example is the application of the emotional models as part of the mechanisms that construct the stories and quests in the video games. We will here present a character, controlled by the human player, that will have to complete a scenario trying to enter into a well-guarded castle. We will finally illustrate two different approaches that can be used to build this kind of quests, the traditional milestone approach and the emotion-driven mechanism.

■

[3]http://www.bioware.com/games/legacy and http://nwn.wikia.com/wiki/Main_Page

4.3 Emotional Elicitation Process

The **Emotional Elicitation Process (EEP)** model is an independent architecture for synthetic emotions simulation and can act as Emotional Engine in the AGCBAR Architecture, but it can also be applied as a standalone emotional model for some other simulation environments and architectures. In this section we will describe the inputs and outputs that the EEP model requires. This model encloses all the procedures that are necessary to evaluate the emotions and to manage the mood state and dynamics, according to the environment and the character, see Figure 4.2. The main attributions of this model are: (1) It has simple mechanisms that grant initial emotional characteristics to agents, actions and events. (2) It provides a new structured analysis of the events, according to the widely admitted OCC model [60], while based also on the PAD space [51]; all this allows it to project, quantify and manipulate the emotions numerically (the EEP provides both, the mechanism to compose the emotions prompted by each particular event, and the capability to update the mood of the character). (3) It projects the character's mood into a Mood Vector Space, that is modeled as a formal mathematical tridimensional space, with enough functions to grant the correct combination, measure and decay of the emotions that affect the mood. (4) Last, it has a clear interface that makes possible the integration of the EEP model as a video game component.

4.3.1 Architecture Overview

The EEP model is divided into four components, as shown in Figure 4.2:

- **Conceptual Dictionaries (CD)**: they provide the definitions of the general elements, needed by the engine to evaluate the events. For example, the type of Actions that the event could record. These dictionaries are different from the AGCBAR Dictionaries and they are only related with the internal procedures of the EEP model.

- **Character Profile (CP)**: it defines the numerical parameters that represent the the general definition of the personality of the character and the identification of the *Moods State*, μ_i, in the mood space, $\mathcal{M}$. It also provides the quantification of the elements recorded by the Conceptual Dictionaries. For example, the value of praiseworthiness that a particular character associates to the Action of Give-Money recorded in the dictionaries.

- **Emotional Elicitation Process Engine (EEPE)**: it decomposes, analyzes and evaluates an event. It also analyzes the emotions and their intensity levels as result of the evaluation of an event E_i.

- **Mood Vector Space (MVS)**: it represents the current Mood State of the character, μ_i. The MVS is a 3-Dimensional space derived from the PAD model [51]. Here, the emotions perceived by a character are transformed by the EEPE into vectors in this

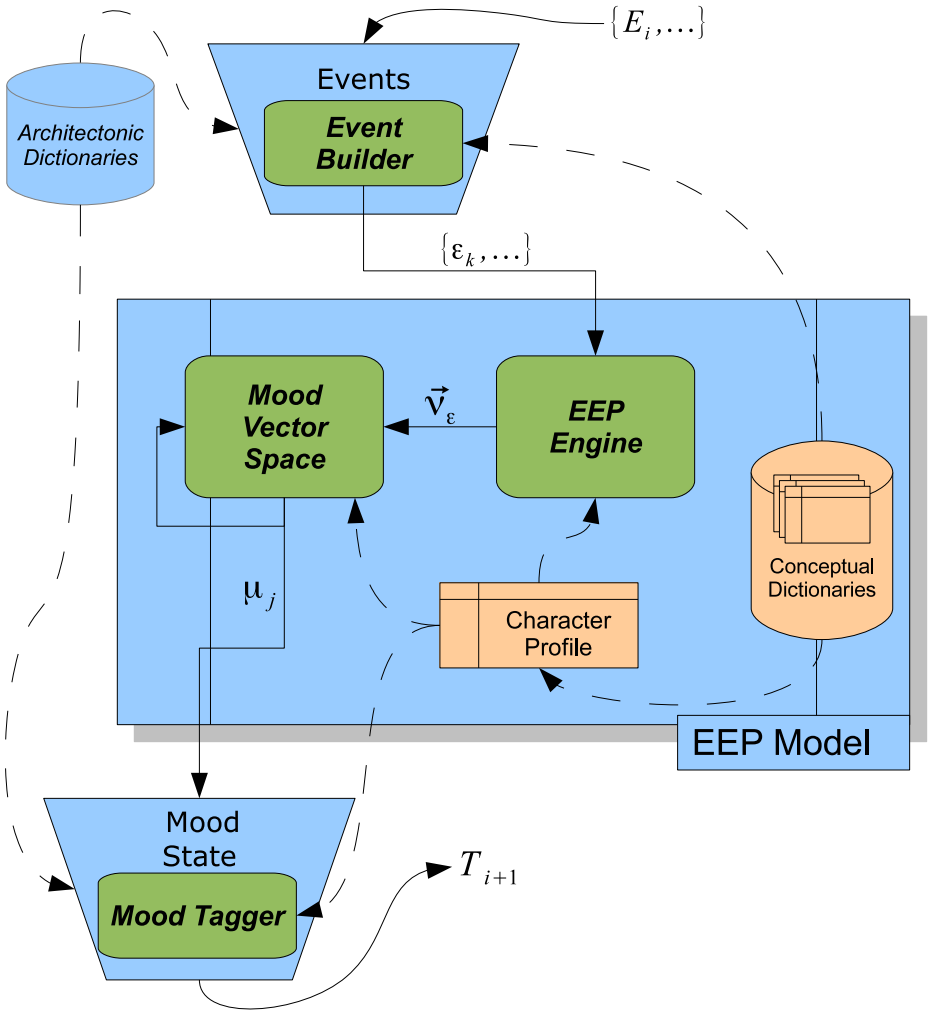


Figure 4.2: EEP General Architecture. The EEP Model encloses the EEP Engine and the Mood Vector Space for each of the characters to produce the *Mood Dynamics* according to their specific profiles. The Event Builder adapts the emotional events produced in the Game Engine to the representation and information of the events that the EEP Model needs.

space, and they translate the character's mood state from one point to another within the boundaries of the MVS. For example, a concrete point of this space can represent a mood state for a particular character at a particular given time.

Therefore, outside of the model, we can implement the interfaces that synthesize the events generated by the environment (Game Engine in the case of the AGCBAR Architecture) so they can produce the moods prompted the Emotional Engine:

- **Event Builder (EB)**: given the events generated by the environment, E_i, this component adapts itself to the structure needed by the EEPE. This component can be defined as a *wrapper* of incoming events. Each incoming event may produce 0 to N internal EEP events, $\{\varepsilon_j\} \in \mathcal{E}$, (because of the causal relationship among the events and their components).

- **Mood Tagger (MT)**: the Moods are a discrete set of finite values that have their meaning in the context of the Game Engine. However, the EEP Model operates on a continuous representation of the Mood State, which is based on the MVS representation. It is the responsibility of the MT to project the continuous mood state, $\mu_i \in \mathcal{M}$, to one of the discrete values, $T_i \in \mathbf{T}$. This state space is the basics representation from which we extract the Mood State, so we can translate it later into the Mood Tags $T_i \in \mathbf{T}$ through the Mood Interface in the AGCBAR Architecture.

$$
\begin{array}{cccccc}
\text{Event} & & \text{Event Classes} & & \text{Mood} & \text{Mood Tags} \\
E_i & \in & \mathbf{E} & & T_j & \in \quad \mathbf{T} \\
\downarrow EB & & & & \uparrow MT & \\
\{\varepsilon_1, \ldots, \varepsilon_n\} & \subset & \mathcal{E} & \xrightarrow[EEPE]{} & \mu_j & \in \quad \mathcal{M} \\
\text{EEP Events} & & \text{EEP Event Classes} & & \text{Mood State} & \text{MVS}
\end{array}
$$

This augmented structure of events is necessary for the balanced analysis of the emotions. It also provides the parameters that are crucial for the classifying and quantifying the emotions produced by the events. For example, the events are associated according to their sources and targets, but also their causal consequences are translated as emotional events.

In addition to that, a continuous representation of the Mood State allows the EEP Engine to operate with fine grain transitions (prompted by the events). It also allows the projection into the discrete number of mood tags defined by the AGCBAR Architecture.

4.3.2 Architecture Dynamic View

All these elements are part of a structured process that continuously updates the mood states of the characters. This process takes into account the input from the events

happening in the environment and their temporal dimension to propagate and cause the decay of each mood state. This procedure has the following general steps:

1. The events, E_i, produced by the changes in the environment are processed at the EB and after that, they generate a set of augmented events, ε_k. Each event has the following components: (1) the source of the event (animate or inanimate), (2) the target of the event (animate, inanimate or none), (3) the consequences of this event, (4) the actions that produce these consequences and (5) the objects related to this event.

2. Using the configurations recorded in the CP, the EEPE generates a vector representation of the emotions produced by an event, $\overrightarrow{\nu_\varepsilon}$. In order for this to happen, the EEPE operates in two phases:

   (a) First, it decomposes and analyzes the event. The engine then produces a set of emotions, according to the structured analysis of the event.

   (b) Second, these emotions, elicited in the EEPE, are translated (also by the EEPE) into the scaled vectors of the MVS (described with the PAD components [51]).

3. These "vectorized" emotions (the representation of the emotions quantified and represented as vectors) are composed and added to the current mood state $\mu_i \in \mathcal{M}$ according to the MVS operators.

4. The MT provides the mechanisms to find the projected mood tag, $T_i \in \mathbf{T}$ according to the parameters stored in the CP. This mechanism enables the communication with the AGCBAR Architecture (or with any other component that needs the current mood of a character to perform further actions).

This model is part of the individual control mechanism of a virtual character. Thus, each character has an instance of the EEP Model associated in order to produce the emotional and mood simulations in his behavior, (see Figure 4.3). Therefore, when an event, E_i is produced in the environment, those characters that perceive it experience an alteration in their mood state when their instance of the EEP Model processes the event.

4.3.3 Mood Vector Space

The Mood Vector Space (MVS) component of the architecture provides the mathematical representation that manages the initial mood states, the state transitions and the emotion effects. The MVS ensures the robustness of the transitions and the mood representation, and it also supports the projection of mood states into the appropriate mood tags. Relevant operations and structures of the MVS, and a sample mood space (the trigonometric mood space) are explained in detail in the Appendix B.1.

Given the fact that the MVS is a *three dimensional space for temperament and mood representations*, the space is built with the following different objectives:

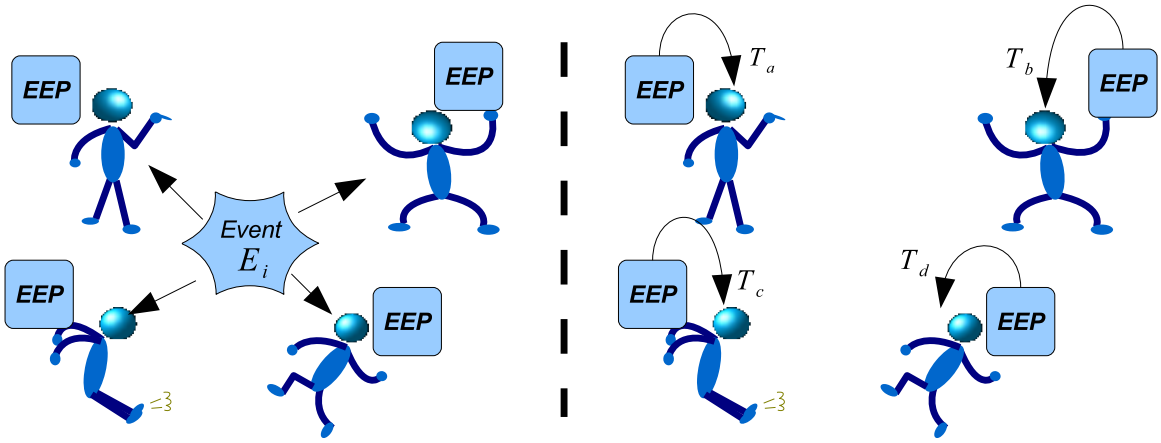


Figure 4.3: EEP Instances per Character. Each character has his own instance of the EEP Model so he can update his mood state according to the events perceived.

- to compose emotions with the current mood state. This could be seen as a translation (composition) in the 3D bounded space,
- to apply the function that represents the decay of the emotions though a time period that drives the character's mood state toward the default state (derived from his particular personality traits),
- to define the distance metric from and to the different reference points in order for it to assign the closest Mood Tag for the current mood state to one of them.

Thus, Mood Vector Space is defined as a tuple $MVS =< \mathcal{M}, \oplus, \cdot, \ominus, \| \cdot \|, \mathfrak{A} >$ where:

- $\mathcal{M} = [-1, 1]^3 \subset \mathbb{R}^3$ is a bounded 3-Dimensional continuous space to represent either moods or emotions,
- $\oplus$ is the operation that composes and translates the emotional vectors in the Mood Vector Space,
- $\cdot$ is the scalar product that allows to scale emotional vectors according to the intensity parameters,
- $\ominus$ and $\| \cdot \|$ are the operations that calculate the distances in the Mood Vector Space.
- $\mathfrak{A}$ is a family of attenuation functions. Each of these functions drives the current mood μ_i back along the time toward its initial state μ_0

The space of representation of the mood is modeled by a bounded $\mathcal{M} \subset \mathbb{R}^3 \mid \mathcal{M} = [-1, 1]^3$ space that corresponds to the 3-axed orthogonal space constructed by the use of the three PAD components (Pleasure-Arousal-Dominance). This framework enables the

transition between moods using the emotions. It is possible, by treating the emotions as vectors, to translate the current mood position into the MVS.

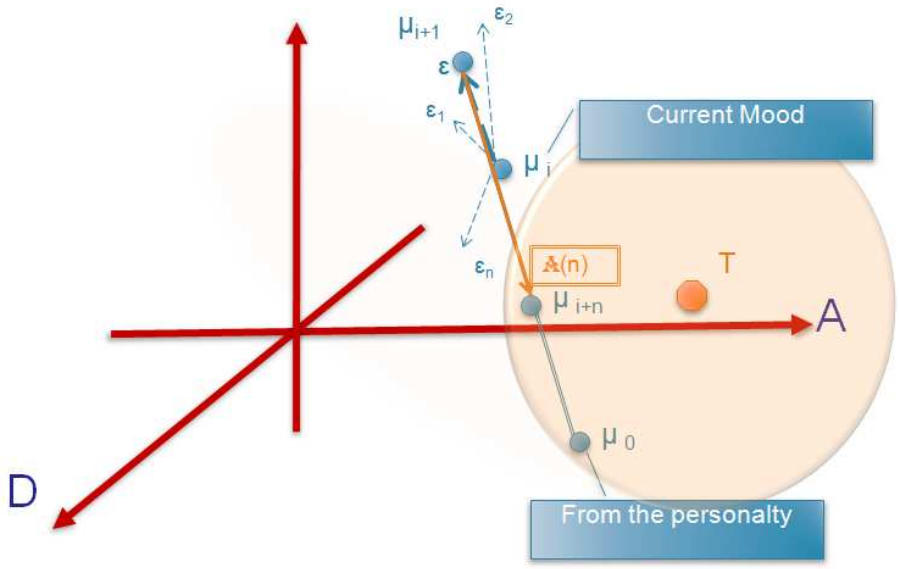


Figure 4.4: MVS example. μ_i transits to μ_{i+1} according to the composition of emotions $\{\varepsilon_j\}$. After n units of time, the mood is decaying toward μ_0 applying the attenuation function $\mathfrak{A}$ (personality baseline mood) up to the μ_{i+n} position.

4.3.4 Conceptual Dictionaries

As we said, the Event Builder takes the emotional events, E_i, produced at the Game Engine, and adapts them to the format and needs of the EEP Model. To be able to do so, we need to align the events with the character's profile. Therefore, it is necessary a set of repositories (dictionaries) that store all the relevant elements that could appear in the game environment in which we want to apply the EEP model. These dictionaries are, in some cases, extensions of the AGCBAR Architecture Dictionaries. We will differentiate here those five different dictionaries:

1. *Consequences Dictionary* ($D^c = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$): the Consequences are the character's expectations about things that could happen independently of any belief about their possible causes. In a combat situation it could be as simple as all the possible (short term / long term) results (for the character himself and for his team and allies), such as been attacked, hit or killed. In a storytelling application these consequences basically related with story plot milestones (see Table 4.2-a).

2. *Actions Dictionary* ($D^a = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$): the Actions represent what the characters can do within the environment. For instance, in a combat scenario these are

attacking, defending or any other relevant movement actions. The actions, when using EEP to support storytelling, could be specific dialogue options, certain events performed in the environment, or actions that complete quests (see Table 4.2-b).

3. *Objects Dictionary* ($D^o = \{\omega_1, \omega_2, \ldots, \omega_n\}$): the Objects are physical or conceptual elements that generate liking or disliking emotions by their mere presence on the events. In a combat situation they could be very representative combatants, while, in the case of the storytelling, they could be anything that is relevant for the storyline (gifts, money, quest items, ...) (see Table 4.2-c).

4. *Characters Dictionary* ($D^r = \{\rho_1, \rho_2, \ldots, \rho_n\}$): this dictionary describes the characters that could be present in a particular event. Once a character is involved in an event, his relationship with the perceiving character is evaluated. (see Table 4.2-d).

5. *Groups Dictionary* ($D^g = \{g_1, g_2, \ldots, g_n\}$): the characters of the Characters Dictionary can belong to a set of groups that can be used to identify general relationships with a specific set of characters. So, $g_i \subseteq D^r$ could appear in the scenario (see Table 4.2-e) as a set of characters conceptually grouped. The perceiving character can not have a specific relationship with the characters involved in the events, but can have a general relationship with some of the groups they belong to. These group relationships can affect in the same way that the character specific relationships do.

As in the case of the AGCBAR Architecture dictionaries, the Conceptual Dictionaries, proposed in the EEP Model, will be used to ensure the alignment between the characters' profiles (configurations) and the rest of the components of this model. They enclose the definitions of the different parameters representing the environment and the emotional elements. With these parameters, the model recognizes the components of the inputs (emotional events) and identifies their influence in the mood state dynamics of the characters.

The ***combat scenario*** example involves the attack and defensive actions and the consequences of these actions for the evaluation of characters' emotional responses. The relationships of the characters in the groups of combatants determine the influence of the events perceived by the characters.

The ***storytelling scenario***, addresses the actions produced by the player along the game and the interaction that he has with the characters in the scene. The objects and the actions are tightly related with the advance in the quest. These elements will serve as triggers for the emotional responses from the NPCs.

Continue....

■

....
The following table presents the aforementioned dictionaries specified for each of the proposed scenarios.

	Combat Scenario	**Storytelling Scenario**
(a) Consequences Dictionary	Be-Attacked Be-Hit Be-Killed Hit-Enemy	Talked-Rude Talked-Polite Talked-Neutral Make-Deal
(b) Actions Dictionary	Attack Defend	Talk-Rude Talk-Polite Talk-Neutral Buy
(c) Objects Dictionary	OrcBoss HumanKnight	Kitten Beer Happy-Song Money
(d) Characters Dictionary	OrcArcher HumanArcher OrcWarrior HumanWarrior OrcBoss HumanKnight	HumanPlayer SadGirl BeardedMerchant SkinnyMerchant Bard Watchman
(e) Groups Dictionary	@Orcs @Humans	@Foreigners @FellowCitizens

Table 4.2: Examples of different EEP Conceptual Dictionaries. The dictionaries created for the proposed examples can have, among others, these elements.

4.3.5 Character Profile

The Character Profile (CP) is the specification of the character's preferences and emotional behavior. The profile is composed of three different groups of information: (1) a set of functions that projects for each element detailed in the Conceptual Dictionaries (which will be referred by the events) a value real in the range $[-1, 1]$, (2) the personality traits that describe the general mood state of the character, and (3) the association of the mood space points to mood tags, for those that are relevant for the character.

The CP is formally described as: $CP =< D, P, A, R, \{m^k\}, B >$, where:

$$D : D^c \longrightarrow [-1, 1] \qquad P : D^a \longrightarrow [-1, 1]$$
$$\gamma \longrightarrow d_\gamma \qquad \alpha \longrightarrow p_\alpha$$

$$A : D^o \longrightarrow [-1, 1] \qquad R : D^r \longrightarrow [-1, 1]$$
$$\omega \longrightarrow d_\omega \qquad \rho \longrightarrow r_\rho$$

And:
$m^k \subseteq \mathcal{M}$ are the set of reference points with their corresponding mood T_k, and $B =< o, c, e, a, n >$ is the Big Five personality description composed by the 5 parameters.

Emotional Parameters

The parameters describe the desirability of the consequences, d_γ, the praiseworthiness of the actions, $p\alpha$, the appeal associated to the objects, a_ω, and the relationship with other characters in the scene, r_ρ. These values are selected from the point of view of the character, for example, the Be-Hit consequence must be evaluated as the desirability of being harmed. This will be applied even for the evaluation of the consequences, appealing and praise of perceived third-persons events, as suggested by [60], about the general assumption of external alignment of our own scale of values applied to other persons. All of the emotional parameters must be in one of the Conceptual Dictionaries since any of these elements could appear in an event.

The relationship among characters is slightly different. These relationships can be declared by groups (appending the "@" character) or by individuals, where individual relationship qualification overrides any group value. The resultant relationship with a particular character is calculated by the average of the relationships with the groups in which this character is included.

The different characters, presented in the examples described in this chapter, have a specific set emotional parameters. These parameters are necessary for the EEP Model to operate. For instance, the Human Warrior, appearing in the ***combat scenario***, has the following parameters. The values of these parameters are set within the interval $[-1, +1]$ and they are used for the intensity quantification of the emotions (see Section 4.3.6).

Consequence Desirability: d_γ	γ_1: Be-Attacked	-0.3
	γ_2: Be-Hit	-0.5
	γ_3: Be-Killed	-0.9
	γ_4: Hit-Enemy	0.2
	γ_5: Kill-Enemy	0.6
	γ_6: Fled	-0.1
Action Praiseworthy: p_α	α_1: Attack	0.04
	α_2: Defend	-0.1
	α_3: Power-Attack	0.1
	α_4: Move-Away	-0.3
Object Appealing: a_ω	ω_1: OrcBoss	-0.2
	ω_2: HumanBoss	0.1
Relationships: r_ρ	ρ_1: HumanArcher	0.5
	ρ_2: HumanKnight	0.8
	ρ_3: @Orcs	-0.9

Table 4.3: Example of the EEP Emotional Parameters for one of the characters in the Combat Scenario

Personality Traits

The personality description is based on the Big Five Factors, as shown in Section 4.2.2. This representation of the personality is widely used in synthetic models for virtual characters [88, 14]. The five parameters of this representation are associated with different aspects of the personality of a person. To implement them correctly we describe each of these parameters with a value in the range of $[-1, 1]$. The values of these parameters must be projected into the MVS (as a PAD 3D representation) in order to be able to compute the transitions and the default mood point derived by the character personality. The transformation is taken from the equations proposed by Mehrabian [50]. These equations are easy to compute as shown in Table 4.4.

These equations will be addressed in the EEP Engine Algorithm as $B5toPAD$ functions.

$$B5toPAD : [-1, 1]^5 \longrightarrow \mathcal{M}$$
$$< o, c, e, a, n > \longrightarrow < p, a, d >$$

The value obtained by this function is the representation of the initial and central mood state of the character, $\mu_0 \in \mathcal{M}$ (projection of Mood T_0), which will be used as starting and attracting point of the mood state that the character will have during the simulation.

$\mathbf{P}leasure =$	$0.21 \cdot Extraversion + 0.59 \cdot Agreeableness$
	$+0.19 \cdot Neuroticism$
$\mathbf{A}rousal =$	$0.15 \cdot Openness + 0.30 \cdot Agreeableness$
	$+0.57 \cdot Neuroticism$
$\mathbf{D}ominance =$	$0.25 \cdot Openness + 0.17 \cdot Conscientiousness$
	$+0.60 \cdot Extraversion - 0.32 \cdot Agreeableness$

Table 4.4: Mehrabian's Big Five to PAD transformation rules [50]. These equations transform the Big Five personality parameters into the PAD components.

In our combat scenario example, we designed the personality of the human archer as cautious, extrovert and friendly. These traits represent his temperament, applying general personality models (according to the definition and semantic of the parameters), as:

Openness:	0.4	*Conscientiousness*:	-0.1	*Extraversion*:	0.4
Agreebleness:	0.5	*Neuroticism*:	-0.1		

These personality values produce the corresponding initial mood state value, μ_0, in terms of the PAD representation of:

Pleasure:	0.36	*Arousal*:	0.153	*Dominance*:	0.163
$\mu_0 = (0.36, 0.153, 0.163)$					

■

Projecting Mood to Mood States in MVS

As we will see later, the combination of the emotions, prompted by the events, will change the current Mood State of the character in the MVS giving a unique position at a given timestamp.

It is the responsibility of the MT to project all $\mu_i \in \mathcal{M}$ into the corresponding Mood $T_k \in \mathbf{T}$. Although there are multiple alternatives for doing this, we selected the approach of assigning a subset $m^k = \{\mu_1^k, \ldots, \mu_n^k\} \subseteq \mathcal{M}$ of reference points to each T_k. The MT implements the projection function $\mathfrak{L}(\mu_i)$ that selects the T_k mood according to the closest μ_j^k among all the reference points from the different moods.

$$\begin{aligned} \mathfrak{L} : &\mathcal{M} \rightarrow \mathbf{T} \\ &\mu_i \rightarrow T_k \end{aligned}$$

$$\begin{aligned} \mathfrak{L}(\mu_i^k) &= T_k \\ \mathfrak{L}(\mu) &= T_k | \underset{\mu_j \in \bigcup_i m^i}{\arg\min} (||\mu_j \ominus \mu||),\ \mu_j \in m^k \end{aligned}$$

The MVS describes the mechanism that computes the distance between two points in the MVS, $||\cdot||$, being its main objective to identify the closest reference point in the set.

The management of the set of Moods, $\mathbf{T}$, and projection function, $\mathfrak{L}$, may exploit the existence of a mood ontology as the one mentioned in Appendix B.2.

In our combat scenario, we use as guidelines the ontology presented in the Appendix B.2 and the values of Russell & Mehrabian's work [74] in order to allocate three different Mood Tags that represent different characters' moods. These are indicated by a set of reference points, which will be used to conduct different behavior controllers:

T_a =NORMAL $= \{\mu_1 = (0.2, -0.1, 0.3)\}$	T_b =AFRAID $= \{\mu_2 = (-0.2, 0.6, -0.7)\}$
T_c =ANGRY $= \{\mu_3 = (-0.2, 0.8, 0.4),\ \mu_4 = (-0.3, 0.5, 0.1)\}$	

Table 4.5: Example of the identified Mood Tags applied in the Combat Scenario to the Human Archer character

4.3.6 EEP Engine

The EEP Engine is divided in two stages that complete the analysis and projection of an EEP Event, ε, in a vector representation of this event in terms of the emotions that it provokes, ν_ε. The evaluation process can be defined as:

$$\varepsilon \xrightarrow[\text{Event Analysis}]{} \{e_\alpha, e_\gamma, e_{\alpha-\gamma}, e_\omega\} \downarrow \text{MVS projection} \quad \overrightarrow{\nu_\varepsilon}$$

Where we obtain a set of emotions, $\{e_\alpha, e_\gamma, e_{\alpha-\gamma}, e_\omega\}$, produced by the Event Analysis. The MVS Projection these emotions creates a set of MVS vectors that we compose to obtain $\overrightarrow{\nu_\varepsilon}$.

Event Analysis

Once an event ($\varepsilon =< \alpha, \{\gamma\}, \{\omega\}, src, tgt >$) is received by the EEP Engine the general sequence of evaluation begins.

Where an EEP event:

$$\varepsilon_i =< \alpha, \gamma, \omega, src, tgt >$$

is composed by, α, an action of those described in the Actions Conceptual Dictionary D^a, $\alpha \in D^a$, γ, the set of consequences $\gamma_i \in D^c$ of the event, ω, the set of the objects $\omega_i \in D^o$ involved in the event and, src and tgt, the identifiers of the involved characters (source and target) of the event from D^r.

This evaluation is based on the valence analysis of emotions of Ortony, et al.[60].

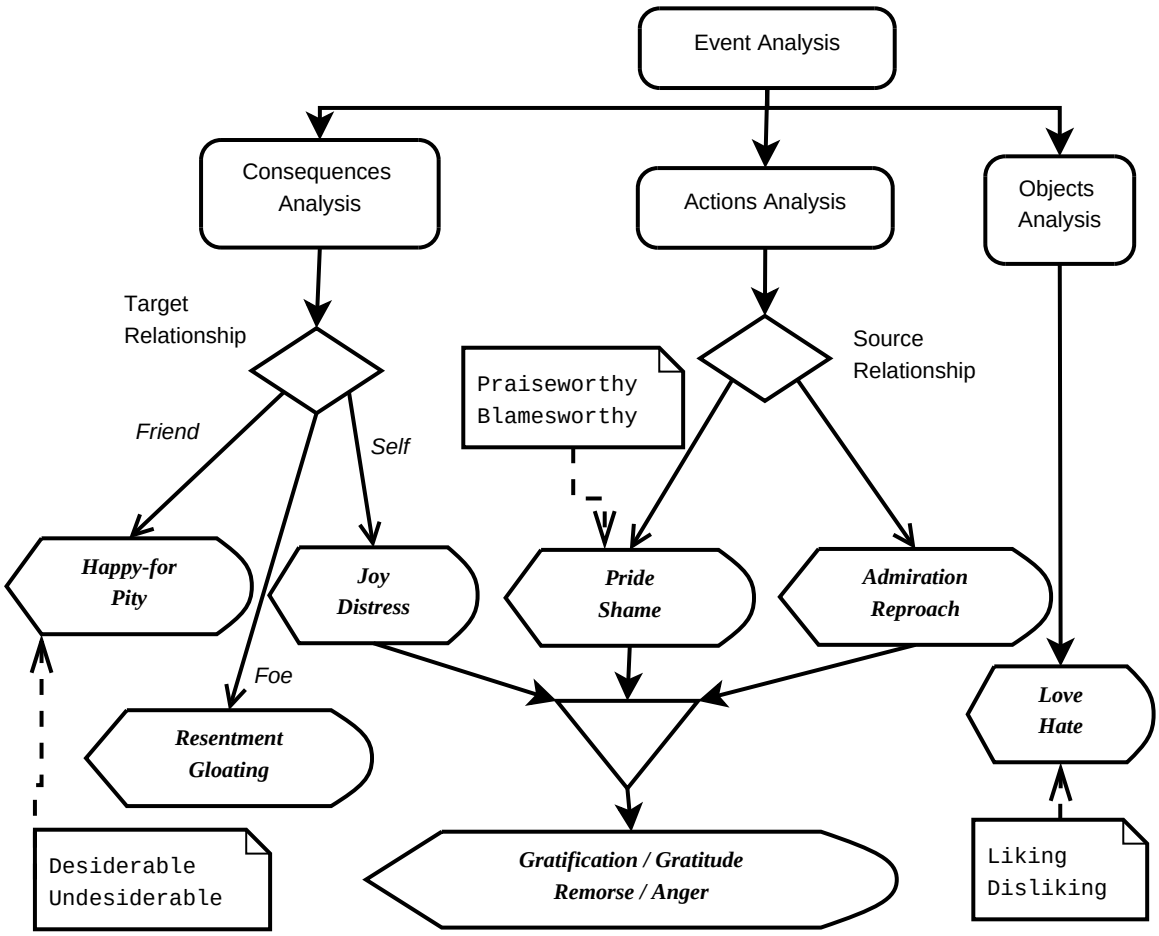


Figure 4.5: EEP Engine Event Evaluation. Based on the OCC model classification of emotions. The emotions $e_i \in \mathbb{E}$ are extracted according to the event, ε_i components (Consequences, Actions and/or Objects).

For all the sequence of analysis we classify the relationship among the characters

as <*Self, Friend, Foe*>. According to the relationship of the character with the source and/or target of the event, we describe *Friend* with the relationship quantified by values "> 0" and *Foe* in the cases of relationships "< 0". The value of the relationship, $r \in [-1, 1]$, is calculated through the Relationship and Group Dictionaries.

The evaluation sequence is as follows:

1. The evaluation of the emotion, $e_\alpha \in \mathbb{A} \subset \mathbb{E}$, produced by the Action, $\alpha \in D^a$, of the event perceived, is done by the function: $action(\alpha, p_\alpha, r_{src}) = e_\alpha \in \mathbb{A}$,

$$action : D^a \times [-1, 1] \times [-1, 1] \longrightarrow \mathbb{A}$$
$$(\alpha, p_\alpha, r_{src}) \xrightarrow{action} e_\alpha$$

   where $p_\alpha \in [-1, 1]$ is the praiseworthiness of the action α, and r_{src} is the relationship between the perceiving character and the source of the action. The evaluation is done as explained in Table 4.6. This evaluation takes its value from the set of `Attribution` Emotions $\mathbb{A} = \{$Pride, Shame, Admiration, Reproach$\}$.

Source	$p_\alpha > 0$	$p_\alpha < 0$
Self	Pride	Shame
Friend	Admiration	Reproach

Table 4.6: `Attribution` Emotions produced by the Action α, $e_\alpha = action(\alpha, p_\alpha, r_{src})$. Evaluated according to the action praiseworthiness p_α and the relationship with the source of the action r_{src}

2. Evaluation of the set of emotions, $\{e_{\gamma_i}\} \subset \mathbb{C} \subset \mathbb{E}$, produced by the Consequences ($\gamma_i \in D^c$) of the event perceived: $conseq(\gamma_i, d_{\gamma_i}, r_{tgt}) = e_{\gamma_i} \in \mathbb{C}$

$$conseq : D^c \times [-1, 1] \times [-1, 1] \longrightarrow \mathbb{C}$$
$$(\gamma_i, d_{\gamma_i}, r_{tgt}) \xrightarrow{conseq} e_{\gamma_i}$$

   where $d_{\gamma_i} \in [-1, 1]$ is the desirability of the consequence $\gamma_i \in D^c$. The consequences can generate the `Well-Being` and `Fortune-of-Others` emotions, $\mathbb{C} = \{$Joy, Distress, Happy-For, Pitty, Resentment, Gloating$\}$, according to the analysis described in Table 4.7.

3. When the Actions and Consequences of an event prompt certain emotions, the combination of these emotions produces another set of emotions, called `Well-Being/Attribution Compound` emotions, $\{e_{alpha-\gamma_i}\} \in \mathbb{T} \subset \mathbb{E}$: $attr(e_\alpha, e_{\gamma_i}) = e_{alpha-\gamma_i} \in \mathbb{T}$, where $e_\alpha \in \mathbb{A}$, $e_{\gamma_i} \in \mathbb{C}$.

$$attr : \mathbb{A} \times \mathbb{C} \longrightarrow \mathbb{T}$$
$$(e_\alpha, e_{\gamma_i}) \xrightarrow{attr} e_{\alpha-\gamma_i}$$

Target	$d_\gamma > 0$	$d_\gamma < 0$
Self	JOY	DISTRESS
Friend	HAPPY-FOR	PITY
Foe	RESENTMENT	GLOATING

Table 4.7: `Well-Being` and `Fortune-of-Others` Emotions produced by the Consequences γ_i, $e_{\gamma_i} = conseq(\gamma_i, d_{\gamma_i}, r_{tgt})$. Evaluated according to the consequence desirability d_{γ_i} and the relationship with the target of the action r_{tgt} values

The combination is explained in Table 4.8. The Attribution emotions are only produced by the self-consequences of the event, therefore, the only e_{γ_i} that produce Attribution are JOY and DISTRESS[60].

e_α	e_{γ_i} =JOY	e_{γ_i} =DISTRESS
PRIDE	GRATIFICATION	
SHAME		REMORSE
ADMIRATION	GRATITUDE	
REPROACH		ANGER

Table 4.8: `Compound` Emotions produced by the combination of the emotions e_α and e_{γ_i}, $e_{\alpha-\gamma_i} = attr(e_\alpha, e_{\gamma_i})$

4. Evaluation of the emotions, $\{e_{\omega_i}\} \in \mathbb{O} \subset E$, produced by the Objects ($\omega_i \in D^o$) of the event: $object(\omega_i, a_{\omega_i}) = e_{\omega_i} \in \mathbb{O}$

$$object : D^o \times [-1, 1] \longrightarrow \mathbb{O}$$
$$(\omega_i, a_{\omega_i}) \xrightarrow{object} e_{\omega_i}$$

where $a_{\omega_i} \in [-1, 1]$ is the appeal of the object $\omega \in D^o$. The evaluations are made according to Table 4.9 producing the `Attraction` Emotions, $\mathbb{O} = \{$LOVE, HATE$\}$.

$a_\omega > 0$	$a_\omega < 0$
LOVE	HATE

Table 4.9: `Attraction` Emotions produced by the Object ω_i,$e_{\omega_i} = object(\omega_i, a_{\omega_i})$. Evaluted according to the appeal of the object a_{ω_i}

MVS Projection

Given the set of emotions $\{e_i\} \in \mathbb{E}$, produced by the event $\varepsilon =< \alpha, \{\gamma\}, \{\omega\}, src, tgt >$, we proceed to project those tagged emotions into the MVS. Therefore, we transform each

emotion, e_i, into a vector representation in the MVS, $\overrightarrow{\nu_i} \in \mathcal{M}$, through the function $pad(e) = \overrightarrow{\nu}$.

$$pad : \mathbb{E} \longrightarrow \mathcal{M}$$
$$(e_i) \longrightarrow \vec{\nu_i}$$

This function is implemented with the equivalence shown in Table 4.10 given the representation of an emotion that we interpret as a vector that will move the character's mood toward that point in the MVS.

Emotion	**P**	**A**	**D**	**Mood Octant**	
ADMIRATION	0.50	0.30	-0.20	+P+A-D	Dependent
ANGER	-0.51	0.59	0.25	-P+A+D	Hostile
DISTRESS	-0.40	-0.20	-0.50	-P-A-D	Bored
GLOATING	0.30	-0.30	-0.10	+P-A-D	Docile
GRATIFICATION	0.60	0.50	0.40	+P+A+D	Exuberant
GRATITUDE	0.40	0.20	-0.30	+P+A-D	Dependent
HAPPYFOR	0.40	0.20	0.20	+P+A+D	Exuberant
HATE	-0.60	0.60	0.30	-P+A+D	Hostile
JOY	0.40	0.20	0.10	+P+A+D	Exuberant
LOVE	0.30	0.10	0.20	+P+A+D	Exuberant
PITY	-0.40	-0.20	-0.50	-P-A-D	Bored
PRIDE	0.40	0.30	0.30	+P+A+D	Exuberant
REMORSE	-0.30	0.10	-0.60	-P+A-D	Anxious
REPROACH	-0.30	-0.10	0.40	-P-A+D	Disdainful
RESENTMENT	-0.20	-0.30	-0.20	-P-A-D	Bored
SHAME	-0.30	0.10	-0.60	-P+A-D	Anxious

Table 4.10: OCC to PAD Mapping. Values of the emotions in the PAD representation and of the octant that includes them.

If we look at one particular instant of the ***combat scenario*** example, and take the point of view of the *Human Warrior*, one possible event in the scenario could be the successful attack of the *Orc Boss* to the *Human Knight*. Then the event perceived by the *Human Warrior* would be:

$\varepsilon =$<ATTACK, {BE-HIT, BE-ATTAKED},$\emptyset$,*OrcBoss*,*HumanKnight*>

Continue....

.... According to the parameters presented in the previous example for the *Human Warrior* character we would get:

γ_1: Be-Attacked	-0.3
γ_2: Be-Hit	-0.5
α_1: Attack	0.04
ρ_2: HumanKnight	0.8
ρ_3: @Orcs	-0.9

$$\vec{e_{\gamma_1}} = pad(\text{Pitty}) \cdot \overbrace{0.3}^{|\gamma_1|} \cdot \overbrace{0.8}^{|\rho_2|} = (-0.09, -0.05, -0.12)$$
$$\vec{e_{\gamma_2}} = pad(\text{Pitty}) \cdot 0.5 \cdot 0.8 = (-0.16, -0.08, -0.2)$$
$$\bigtriangledown$$
$$\vec{e_\varepsilon} = \vec{e_\alpha} \oplus \vec{e_{\gamma_j}} \oplus \vec{e_{\alpha\gamma_j}} \oplus \vec{e_{\omega_k}} = (-0.24, -0.13, -0.30)$$

If we analyze the same event, but changing the source and the target:

$\varepsilon =<$Attack, {Be-Hit, Be-Attaked},$\emptyset$,*HumanKnight*,*OrcBoss*$>$

The emotions produced in this case are different:

$$\vec{e_{\gamma_1}} = pad(\text{Gloating}) \cdot 0.3 \cdot 0.9 = (0.08, -0.08, -0.03)$$
$$\vec{e_{\gamma_2}} = pad(\text{Gloating}) \cdot 0.5 \cdot 0.9 = (0.14, -0.14, -0.05)$$
$$\vec{e_\alpha} = pad(\text{Admiration}) \cdot 0.04 \cdot 0.8 = (0,01, 0,01, -0.01)$$
$$\bigtriangledown$$
$$\vec{e_\varepsilon} = \vec{e_\alpha} \oplus \vec{e_{\gamma_j}} \oplus \vec{e_{\alpha\gamma_j}} \oplus \vec{e_{\omega_k}} = (0.22, -0.21, -0,09)$$

■

4.3.7 EEP Event Evaluation Cycle

The EEP Model evaluates the events perceived by the character, according to his Character Profile $CP =< D, P, A, R, \{m^k\}, B >$. As we can see in the Algorithm 2, the Emotional Elicitation Process (EEP) is decomposed as follows:

1. After receiving the initial configuration of the character, we initialize the current mood to its basic state μ_0 applying a translation function $B5toPAD(B) = \mu_0$, where $B \in [-1, 1]^5$ (Big Five Personality Traits) and $\mu_0 \in \mathcal{M}$, where $\mathcal{M}$ is the bound ($[-1, 1]^3 \subset \Re^3$) space of the MVS.

2. We then obtain the emotions $e_i \in \mathbb{E}$ elicited by the event ε, see Figure 4.5, where $\mathbb{E}$ represent all the possible emotions identified in the EEP Model. Being $\varepsilon =< \alpha, \{\gamma\}, \{\omega\}, src, tgt >$, where α is one of the actions described in the Action Dictionary D^a, $\alpha \in D^a$, $\{\gamma\}$ is the set of consequences $\gamma_i \in D^c$ of the event, $\{\omega\}$ is the set of the objects $\omega_i \in D^o$ involved in the event, and src, $tgt \in D^r$ are the identifiers of the source and the target characters of the event. This process is as follows:

   (a) Evaluation of the Action α, that produces the Attribution emotions e_α. We apply the function $e_\alpha = action(\alpha, p_\alpha, r_{src})$ where p_α is the praiseworthiness of the action α and the relationship with the source of the action.

(b) Evaluation of the Consequences, $\{\gamma\}$, enclosing the Fortune-of-Others emotions and the Well-Being emotions, $\{e^i_\gamma\}$. The function $\{e^i_\gamma\} = conseq(\{\gamma_i\}, \{d^i_\gamma\}, \{r^i_{tgt}\})$ process the consequences γ_i, given their desirabilities, d^i_γ, and the relationship, r^i with the targets of these consequences.

(c) Evaluation of the Compound Emotions (Attribution+Well-Being). With the emotions derived from the consequences and the actions, we compose the emotions $\{e^i_{\alpha-\gamma}\} = attr(e_\alpha, \{e^i_\gamma\})$.

(d) Evaluation of the Objects, $\{\omega\}$, that create the Attraction emotions, $\{e_\omega\}$. We obtain these emotions by means of the function $\{e^i_\omega\} = object(\{\omega_i\}, \{a^i_\omega\})$, using the appeal a^i_ω of the objects ω_i.

3. We quantify the emotions prompted by the event, according to the projection of the produced emotions, e_i, into the Mood Vector Space as we detail further in this section: $\forall e \in \mathbb{E}$, $pad(e) = \vec{\nu} \in \mathcal{M}$

4. After we get the vector representation of the emotions, $\vec{\nu}$, on the MVS, we apply the scalar product of the contextual parameters, π_i, (such, distance, relationship strength, etc.) or the profile parameters dependent on the emotion type (p_α, d_{γ_i}, ...). For instance, for the emotions $\overrightarrow{\nu^j_\alpha}$ we scale the vector with priceworthiness, p_α related to this action recorded in the character profile. So:

$$scale(\pi_1, \ldots, \vec{\nu}) = \pi_1 \cdot (\ldots \cdot (\pi_n \cdot \vec{\nu})) = \prod_{i \in [1,n]} \pi_i \cdot \vec{\nu} \in \mathcal{M}$$

5. We compose the scaled emotional vectors, $\vec{\nu}$, produced by the event, ε:

$$\overrightarrow{\nu^j_\varepsilon} = \vec{\nu_\alpha} \oplus \overrightarrow{\nu^j_\gamma} \oplus \overrightarrow{\nu^j_{\alpha-\gamma}} \oplus \overrightarrow{\nu^k_\omega}$$

6. We update the current mood state μ_i, by applying the vector that represents the composition of all of the emotions prompted by the event ν_ε: $\mu_{i+1} = \mu_i \oplus \vec{\nu_\varepsilon}$.

7. We obtain the current Mood Tag according to the projection function, $\mathfrak{L}(\mu i + 1) = T_k$.

In case there is an absence of events, the current mood μ_i tends to move back to the initial mood point derived by the personality μ_0. This is achieved through the application of the attenuation function $\mu_{i+1} = \mathfrak{A}(\mu_i)$.

4.4 EEP Applied to Character Controllers

As we said before, in this chapter we illustrate our explanation of the integration of the EEP Model, by presenting two possible application objectives: First we show a combat

Algorithm 2: EEP Engine Algorithm

Initialization:
begin
Let $CP =< D, P, A, R, \{m^k\}, B >$ the Character's Profile. Where:
D are the desirabilities, P are the praiseworthinesses
A are the appeals, R are the relationships
Each $m_k \subseteq \mathcal{M}$ is the set of reference points corresponding to mood tag T_k
$B =< o, c, e, a, n >$ the Big Five personality description composed by the 5 parameters
$\mu_0 = B5toPAD(B)$, initial mood state
end
begin
if *We Receive Events* **then**
foreach *Event, ε_i, that the EEP receives* **do**
Evaluation of $\varepsilon_i =< \alpha, \{\gamma\}, \{\omega\}, src, tgt >$ where $\{\gamma\}$ is the set consequences of the event, α is the action that triggers the event, and $\{\omega\}$ is the set of objects perceived during the event, and src and tgt are the source and target of the event.
$\overrightarrow{\nu_\alpha} = scale(pad(action(\alpha, p_\alpha, r_{src})), p_\alpha, r_{src})$, scaled vector emotions of the action α
foreach *Consequence γ_j* **do**
$\overrightarrow{\nu_{\gamma_j}} = scale(pad(conseq(\{\gamma_j\}, \{d_{\gamma_j}\}, \{r_{tgt_j}\})), \{d_{\gamma_j}\}, \{r_{tgt_j}\})$, scaled vector emotions of j consequences
$\overrightarrow{\nu_{\alpha-\gamma_j}} = scale(pad(attr(\nu_\alpha, \nu_{\gamma_j}), ||\overrightarrow{\nu_\alpha}||, ||\overrightarrow{\nu_{\gamma_j}}||)$, emotions of well-being/attribution compound
end
foreach *Object ω_k* **do**
$\overrightarrow{\nu_{\omega_k}} = scale(pad(object(\{\omega_k\}, \{a_{\omega_k}\})), \{a_{\omega_k}\})$, emotions prompted by the k objects
end
$\overrightarrow{\nu_\varepsilon} = \overrightarrow{\nu_\alpha} \oplus \overrightarrow{\nu_{\gamma_j}} \oplus \overrightarrow{\nu_{\alpha-\gamma_j}} \oplus \overrightarrow{\nu_{\omega_k}}$
$\mu_{i+1} = \mu_i \oplus \overrightarrow{\nu_\varepsilon}$
end
end
else
//In absence of events
Attenuation of the μ_i along the time
$\mu_{i+1} = \mathfrak{A}(\mu_i)$, apply the attenuation function of the MVS which tends to move the current mood μ_i to μ_0
end
return $T_k = \mathfrak{L}(\mu_{i+1})$ *as the projected Mood Tag*
end

scenario, where the application of the EEP is oriented to the simulation of mood-state-related strategies. And second, (Section 4.5), we exhibit an emotion-driven storytelling example.

Thus, for the selected characters we have modeled their corresponding emotional behaviors:

- Personality traits of the characters: We have designed a unique personality for each character.
- Emotional parameters (for consequences, actions, objects and relationships): Most of these parameters are the same for all the characters in the scenario, or at least for those in the same group (e.g., orcs and humans).
- We have defined three emotional labels for the corresponding Mood Tags NORMAL, ANGRY andAFRAID states (see Table 4.5).
- For each of these emotional labels we have assigned a specific controller designed with Behavior Trees [16]. So the strategies are fixed for each mood state and character.

Besides the design of those controllers, the rest of the parameters represent a minimum configuration effort for all the characters designed for these experiments.

4.4.1 Mood Evolution

We are going to evaluate one of these scenarios in order to trace the emotional evolution of the characters according to the events occurred during a combat in the game. The scenario starts as follows: "*a group of human characters (a fighter, an archer and a knight) are exploring a forbidden temple, when they encounter a gang of raging orcs defending one of the chambers in the temple. This gang is led by an orc boss. When the human explorers enter that chamber, the orcs charge upon them, . . . , and the battle begins, . . .*".

As we indicated when we introduced the examples at the beginning of the chapter, we are using the NeverWinter Nights video game to illustrate and guide the game. Through our example, we assumed certain "social" trait that we use to develop the description of the characters. These social attributes are quite common in this kind of fantasy worlds:

1. we assume that the orcs are savage warriors, with a predilection for the fight, so it is perfectly understandable that they attack as soon as they see the human explorers,
2. the orcs are hot-tempered, so they'll tend to become angry easily,
3. the relationship among the humans is closer (implies more affection feelings) than among the orcs. Thus, the humans tend to worry more about the events in which their acquaintances are involved.

We analyzed the evolution of the sample combat scenario. This scenario has the following main events[4]:

① The two groups are engaged in a combat. One of the Orcs attacks the *Knight*, the other *Orc* and the *Orc Boss* attack the *Fighter*.

② The *Orc Boss* suffers some damage because of the *Fighter*'s counterattack.

③ The *Fighter* kills one of the Orcs but, right after that, he is slayed by the *Orc Boss*.

④ The *Orc Boss* joins the other *Orc* in his combat with the *Knight*.

⑤ The *Knight* dies and the *Orc* and the *Orc Boss* advance to chase the *Human Archer*.

When analyzing some of the actions presented, we find two interesting emotional responses:

- *Orc Boss*: Has a short-tempered personality, $BigFive = (-0.5, -0.5, 0.1, 0.3, -0.4) \Rightarrow pad(\mu_0) = (-0.217, -0.039, 0.035)$[5] in the DISDAINFUL octant according to Table 2.1.

  1. The moment ①, the *Orc Boss* attacks and damages the *Fighter*. Then the EB produces the following events:

  $$\varepsilon =< ATTACK, BE-ATTACKED, BE-HIT, \emptyset, OrcBoss, HumanWarrior >$$

  2. This character (*Orc Boss*) has 0.5 praiseworthiness for the attacking action, that brings out its PRIDE emotion as well as its 0.6 desirability of the consequence of hitting the enemy that drives the emotion of JOY.

  $$\begin{aligned}\overrightarrow{e_\alpha} &= scale(pad(\text{PRIDE}), 0.5)\\ &= scale([0.4, 0.3, 0.3], 0.5) = [0.2, 0.15, 0.15]\\ \overrightarrow{e_{\gamma 1}} &= scale(pad(\text{JOY}), 0.5)\\ &= scale([0.4, 0.2, 0.1], 0.5) = [0.2, 0.1, 0.05]\\ \overrightarrow{e_{\gamma 2}} &= scale(pad(\text{JOY}), 0.5)\\ &= [0.2, 0.1, 0.05]\end{aligned}$$

  3. The action/consequence pair not only prompts PRIDE and JOY, but also produces a GRATIFICATION emotion.

[4]The video and the detailed log that traces this particular scenario can be downloaded from http://www.ia.urjc.es/∼luispenya/research/eep/

[5]Applied the B5toPAD function described in the Section4.3.5.

$$\begin{aligned}
\overrightarrow{e_{\alpha-\gamma_1}} &= scale(pad(\text{GRATIFICATION}), ||\overrightarrow{e_\alpha}||, ||\overrightarrow{e_{\gamma_1}}||) \\
&= scale([0.6, 0.5, 0.4], |||[0.2, 0.15, 0.15]|||, |||[0.2, 0.1, 0.05]|||), \text{ applying B.20} \\
&= scale([0.6, 0.5, 0.4], 0.45, 0.35) \\
&= [0.094, 0.078, 0.063] \\
\overrightarrow{e_{\alpha-\gamma_2}} &= scale(pad(\text{GRATIFICATION}), ||\overrightarrow{e_\alpha}||, ||\overrightarrow{e_{\gamma_2}}||) \\
&= [0.094, 0.078, 0.063]
\end{aligned}$$

The combination of these emotions produces the aggregated emotion vector $\vec{e_\varepsilon}$:

$$\begin{aligned}
\vec{e_\varepsilon} &= \overrightarrow{e_\alpha} \oplus \overrightarrow{e_{\gamma_1}} \oplus \overrightarrow{e_{\gamma_2}} \oplus \overrightarrow{e_{\alpha-\gamma_1}} \oplus \overrightarrow{e_{\alpha-\gamma_2}} \\
&= [0.58, 0.43, 0.34], \text{ applying B.13}
\end{aligned}$$

4. The combination of these emotions, (the $\vec{e_\varepsilon}$ calculated previously), together with some other actions performed by the Orcs, sets the mood state of the *Orc Boss* to $\mu_1 = (0.562, 0.555, 0.507)$.
$$\mu_1 = \mu_0 \oplus \vec{e_\varepsilon} \oplus \dots$$
5. The *Orc Boss* keeps attacking and damaging the *Fighter*, but it is also injured in the instant ②. Therefore, the *Orc Boss* reaches the mood $\mu_2 = (0.764, 0.741, 0.667)$.
6. When reaching this mood state, the point in the PAD space is closer to the ANGRY Mood Tag than to the NORMAL Mood Tag, thus the *Orc Boss* is enraged (changing its behavior, see round 2 in the Figure 4.6[6]).

$$\begin{aligned}
\mathfrak{L}(\mu_0) &= \text{NORMAL} \Leftarrow \underset{\mu_j \in \bigcup_i \mu^{T_i}}{\arg\min} (||\mu_j \ominus \mu_0||) \\
\mathfrak{L}(\mu_1) &= \text{NORMAL} \\
\mathfrak{L}(\mu_2) &= \text{ANGRY}
\end{aligned}$$

- *Human Archer*: He has a completely different personality,

  $BigFive = (0.5, -0.1, 0.4, -0.1, 0.4) \Rightarrow pad(\mu_0) = (0.360, 0.153, 0.163)$

  in the EXUBERANT octant according to Table 2.1. His mood evolution during the scenario is shown in the Figure 4.7.

  1. At instant ②, and because of the events happening during the attack, the *Human Archer* reaches a mood state $\mu_1 = (0.342, 0.164, 0.088)$. This mood state is similar to his initial mood state due to the fact that the *Human Archer* and the *Human Fighter*'s hits, directed to the Orc Archer and the Orc Warrior, produce a small effect (missing the attacks), and also because the attacks received by the human party are not too dangerous either (small damage or misses).

[6]The complete evolution of the values can be found in the Appendices, in the Table B.1

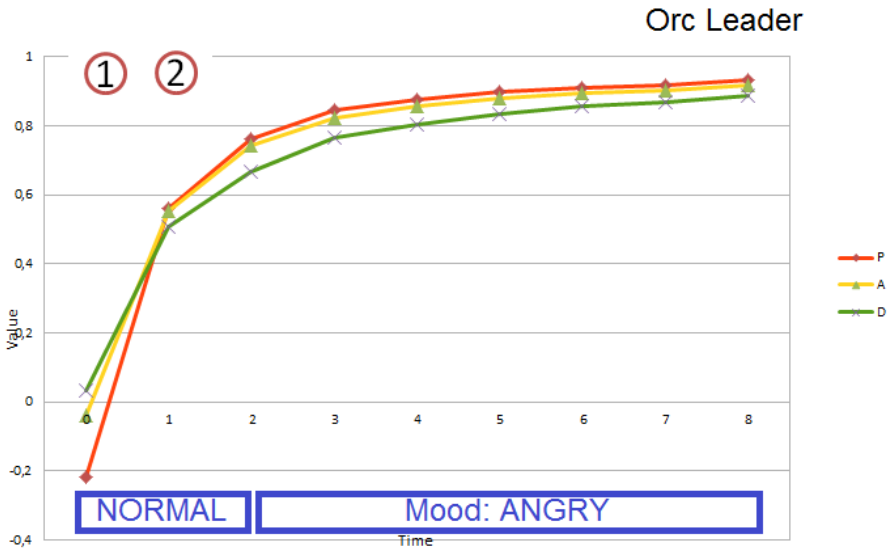


Figure 4.6: Combat Scenario: *Orc Boss* mood evolution along time. ① and ② are key in the mood modification of the *Orc Boss*

2. When his fellow *Human Fighter* dies (instant ③), the *Human Archer* has a desirability $d_{\gamma_1} = -0.9$ associated to the consequence of being killed. As the target of this consequence is a character with a friendship relationship, $\rho = 0.8$, it produces a high intensity PITY emotion. He also perceives the other attacks to his friends and this provokes even more PITTY emotions in him. In addition to that, a consequent counterattack reaction produces some JOY and GRATIFICATION emotions.

$$\begin{aligned}\overrightarrow{e_{\gamma_1}} &= scale(pad(\text{PITTY}), 0.9, 0.8)\\ &= scale([-0.4, -0.2, -0.5])\\ &= [-0.288, -0.144, -0.36]\\ \mu_2 &= \mu_1 + \overrightarrow{e_{\varepsilon_1}} \oplus \overrightarrow{e_{\varepsilon_2}} \oplus \dots\\ &= [0.278, 0.147, -0.0624]\end{aligned}$$

3. At the instant ④, the *Knight* receives attacks from multiple enemies. This situation represents a combined set of emotions in the *Human Archer* of PITY for his friend and RESENTMENT against his enemies. Considering that the *Human Archer* has a strong friendship relationship with the *Knight*, $\rho_3 = 1.0$, all these emotions drop its mood state to the point $\mu_4 = (-0.385, -0.151, -0.670)$ in the PAD space.
4. This point in the PAD space has the Mood Space Point with the AFRAID Mood Tag as the closest label. The *Human Archer* is afraid and tries to defend himself desperately, he also considered the possibility of fleeing.

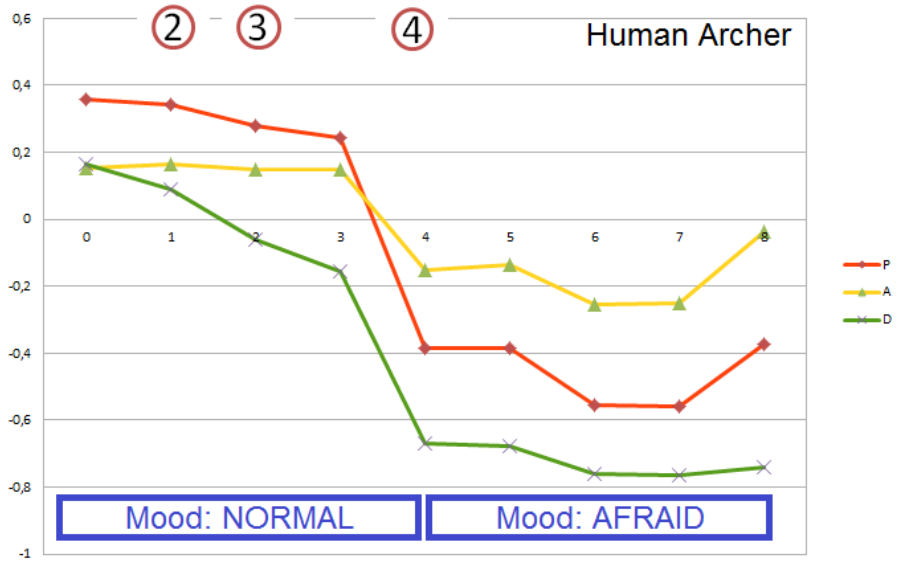


Figure 4.7: Combat Scenario: *Human Archer*'s mood evolution along time. It is drastically marked by the ④ event

4.4.2 Evaluation of the Combat Scenario

As we presented in the introduction of this chapter, the ultimate objective of an emotional model is to make the players believe that the actions of the characters in the game are motivated by some kind of emotion. As we already stated in the discussion of the Chapter 2, the evaluation of the emotion models is still an open issue. In our case, having no available human data on the performance of our given scenarios, we had here to provide the evaluation of the results of such scenarios done by external observers.

To carry out this evaluation we followed this method: (1) First, the tracing of the actions had to be recorded together with the emotional state of the characters, as we explained above. (2) Then, a group of external observers had to judge, according to the behavior and the description of the mood transitions, whether the responses of the characters were believable and whether they matched what it was expected to happen. The questionnaire given to these observers can be found in Appendix B.

The evaluation carried out with this questionnaire is a qualitative one, in terms of psychology research[24], using only *theoretical sampling*, where the limited sampling does not represent a problem because the researcher is not attempting to generalize his findings. We collected the information from a total of 16 expert, regular players. The questionnaire was answered gradually by the experts. First, the experts has to answer the question number 1, then we revealed the next question, presenting more information about the scenario, and so on.

Our questionnaire presents a quantification for the questions from 1 to 5, assuming 1 as a bad qualification in the related subject and 5 a excellent qualification. Therefore,

as overall result, the observers perceived and evaluated the general behavior as coherent (average rate of 4.438±0.814, detailed on Table 4.11). Although, the observers perceive more believability in the behavior of the characters when they were given the information included in the simulation, the personalty of the characters and their emotional parameterization. We think that the first part of the experiment (without giving any information about the emotional engine) was more appropriate to merely evaluate the scenario in terms of the tactical decisions made by the characters. But as we said, when the players knew that there was more complex information behind each of the characters, they found more alignment between the actions and the personalty and emotional profile of the characters.

Thus, the inclusion of the EEP model in the scene made some behaviors that the players did not expect appear. The fact that these behaviors were unexpected was due to the players' previous experiences in this kind of games, where the strategies or behaviors of the characters were not affected by any emotional related event. In the past, the players had usually observed characters that only produced actions not clearly related with their mood, but just based on rational decisions. Hence, the inclusion of an emotional model increases the perception that the characters are not only controlled by fully rational decisions, but also, by certain emotions and moods that could interfere in their good judgment.

	Average Quantification	
Information/Character	*Orc Boss*	*Human Archer*
No Information	4.125 ± 0.619	4.188 ± 0.911
Personality	4.316 ± 0.873	4.500 ± 0.730
Emotional Parameters	4.500 ± 0.730	4.750 ± 0.447
Overall rate of the scenario: 4.438 ± 0.814		

Table 4.11: Combat Scenario: Evaluation results of the questionaire on the character behavior according to the information provided to the evaluator. Questions rated from 1 to 5.

4.5 EEP for Storytelling Support

In this second scenario, the objective is using the EEP model to define storytelling milestones in a more open and emotion-driven way. The traditional storytelling techniques constrain the story progression to a given sequence of actions the player has to perform in a rather restricted order. Moreover, the level designers should consider each quest to be a given sequence of major actions to be achieved to fulfill the story objectives.

In general, the procedure to construct these standard scenarios is based on a series of story plot scenes that contain the major milestones of the quest. Derived by the general implementation design of the video game engines, these story plots are translated into a sequence of paths implemented by a set of scripts or rules that contain the possible pairs of question-answer or event-reaction patterns. These sequences of scripts may be quite restrictive and, in some cases, not coherent because the user might want to interact with

some of the characters in an unexpected, but still reasonable, way to achieve the pursued objective, which might not be considered within the scenario scripts. For instance, the player wants to convince a Non-Playing Character (Non-Player Character (NPC)) to cooperate with him. This can be accomplished by interacting with him in several different ways, and it is only a matter of the NPC's personality, mood state and emotional relationships, to determine if the interactions with the player do or do not produce the desired behavior on the NPC. More precisely, in our scenario the player can gain the collaboration of an NPC bartender in the local inn by either intimidating her, or by helping her to solve a situation with some annoying customers. If this particular scene is configured to be emotion-driven, we have the benefits of: (1) not being a carbon-copy prefabricated pattern but an open interaction-free scenario, (2) the set of actions influencing the target NPC bartender also affects the nearby characters by different means, that produce dynamic and rich playing environments.

4.5.1 Proposed Storyline

This scenario represents a possible quest inspired by many RPG video games, where the player must fulfill certain steps in order to reach a desired goal. In our case, *the player (controlling an epic heroine named Lady Lemma) wants to get in a castle guarded by a watchman that is not willing to allow her to enter. In the village near the castle gate there are different characters (controlled by the AI) that will aid the player to advance in her quest.*

The first explorations of the village will present the following facts:

- The castle guard will not let to Lady Lemma enter through the castle doors.
- Near to the main door of the castle there is a hidden door. But this door is unreachable for the player because it is on the other side of the moat of the castle.
- It seems that there is a pile of crates that can be pushed to build a bridge to cross the moat.
- There is also a melancholic bard sitting on the crates singing a sad song. He is not very cooperative because he wants a beer but the bartender is not willing to give him one.
- In the village market place there are two merchants, a sleeping bartender and a sad girl.

Village Non-Player Characters

In this scenario, there are 6 NPCs that inhabit the village. Each of these characters have his own personality, goals, feelings and relations. The NPCs are shown on Table 4.12.

Sad Girl	She has lost her kitty. She will grant a reward and help the player if she gets her cat back.
Bearded Merchant	He is greedy but willing to bargain.
Skinny Merchant	He is really hard to bargain with. He is envious.
Wandering Bartender	He is carrying his cart selling beer and cheering up the village. Initially he is sleeping.
Melancholic Bard	He is a penniless musician with a very low mood.
Castle Guard	He will not allow to anyone to enter into the castle. He has a very bad temper.

Table 4.12: Storytelling Scenario: List of Non-Playing Characters

For the EEP Model, we need to translate the conceptual design of the characters into the characters' profiles. In the traditional designs, the characters' traits and concepts are retained and discretionarily applied by the designer along the different scripts and dialogues. We include the Sad Girl's profile in Appendix B.

Player Interaction

In this scenario, the player is allowed to interact with the NPCs by means of (1) dialogue options, or by (2) using objects. Furthermore, and in order to clearly represent the different ways to talk with the other characters, the dialogue entries are tagged with three different tags: `Polite`, `Neutral` and `Rude`. Also, when a character uses a certain object, it is reported as an action that usually activates certain conditions on the scenario.

These tags, associated to the dialogue options, are used by part of the scripts to select different branches of the story as representation of the resultant relationship change across the conversational flow. For instance, the *Rude* talking can intimidate or, on the other hand, enrage someone; these selections might make progress the quest in different ways.

4.5.2 General Flow: The Traditional Way

A situation, such as the one explained before, has traditionally been handled by a sequence of expected actions that the player must do. For example, the scenario designer may force the player (controlling Lady Lemma, our heroine) to adhere to the following sequence of steps (see Figure 4.8), in order to find the way to reach the hidden door of the castle:

❶ Lady Lemma must talk with the *Sad Girl*, who will say she has lost her kitty. After this, Lady Lemma must explore the village until she hears a kitty meowing on a house rooftop. She also realizes that she will need a rope in order to climb and reach the rooftop.

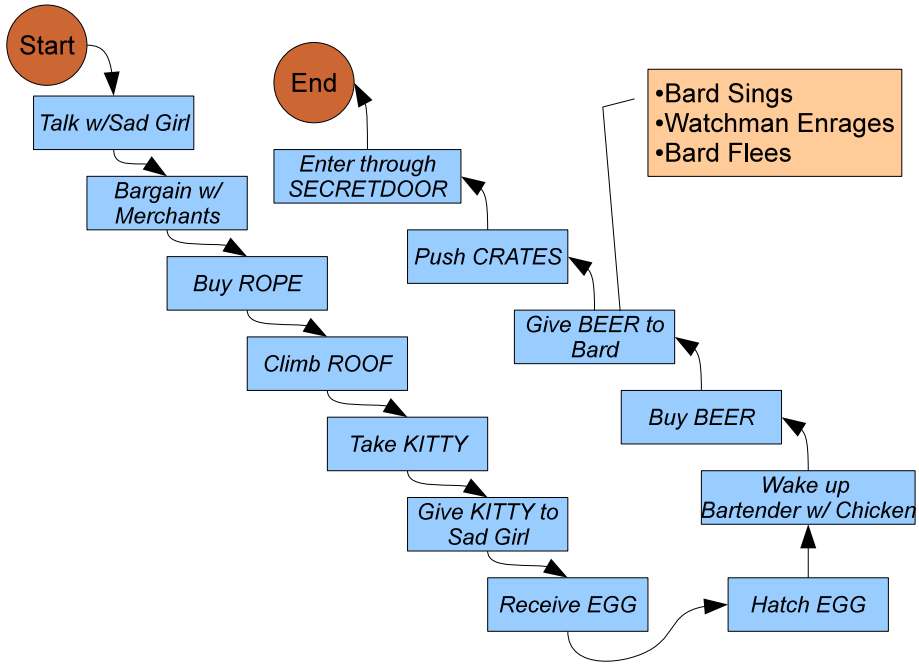


Figure 4.8: Storytelling Scenario: Abstract general plot of the story. The static scripted version describes the sequence of actions that the player is expected to perform in order to fulfill the quest.

❷ Lady Lemma must negotiate with the *Merchants* till they lower the price of the rope, so she can buy one. The bartenders are rivals, so each of them does not want the player to bargain with the other. Lady Lemma must find out which are the best combinations of conversational options to be used in order to obtain the best deal for the rope.

❸ Lady Lemma must climb the roof and get the kitty.

❹ Lady Lemma must bring the kitty back to the *Sad Girl*. The girl then gives her an egg as a gift

❺ Lady Lemma must (somehow) hatch the egg. A chicken comes out of the egg.

❻ Lady Lemma must wake up the *Bartender* (with the help of the chicken that crows).

❼ Lady Lemma must buy a beer from the *Bartender*.

❽ Lady Lemma must give the beer to the *Bard*. The bard becomes cheerful and starts singing. The *Castle Guard* becomes enraged because of the singing and runs after the bard, who flees.

❾ Lady Lemma must now push the empty crates to cross the moat and enters the castle through the secret door.

4.5.3 Emotional Alternative Paths: EEP-based Design

The application of the EEP model aims to exploit the relationships and affects of the characters in order to allow more flexible paths in the story. Based on the complex reactions of the characters, and on the emotions prompted by the actions of the player, the game designer can develop the scenario in a very different way.

In this sense, the design of the different scenes does not need to be defined as a set of strict rules describing the transitions and paths in the story. Instead of that, some of the traditional milestones in the story plot can be described as the necessity of one NPC character to be in certain Mood in order to produce the desired transitions and paths. The mechanisms and options chosen by the player to produce the adequate mood in a character provides many ways to progress in the story. For instance, in Table 4.13, the corresponding steps presented in the previous Section 4.5.2 are translated into a target Mood for a given NPC character out of the possible Moods.

		Mood Tags	
Step	**NPC Character**	Target	Alternatives
❶	*Sad Girl*	NEUTRAL	{HAPPY, SAD}
❷	*Bearded Merchant*	HAPPY	{ANGRY, NEUTRAL}
❷	*Skinny Merchant*	NEUTRAL	{JEALOUS, ANGRY}
❹	*Sad Girl*	HAPPY	{SAD, NEUTRAL}
❽	*Melancholic Bard*	HAPPY	{SAD, AFRAID}
❽	*Castle Guard*	ENRAGED	{SUSPICIOUS, ANGRY}

Table 4.13: Storytelling Scenario: Target Mood Tags for the EEP-Based Design

The actions in the environment (produced by the player, by some NPC, or even by the events of the environment) are perceived by the characters involved in them. These actions produce, as we described in the previous sections, certain changes in the moods of the characters across the scenario. For instance, the good actions (conversational or not) done towards the *Sad Girl* are viewed positively by some characters in the scenario, changing their reactions when they interact with the player. These reaction changes are not only based on the one-on-one interaction with her, but also on the altered good mood that they have produced through the perception of these positive events. This flow of mood states of the characters derived from their relations, personalities and feelings, can produce different paths in the story. Thus, the level designer can tailor a plot based only on the specific moods that certain characters must reach in order for them to move on through the story, leaving the player more freedom to interact with the environment, guided by the emotional reactions of the characters.

Relationships in the Village

If we want to be able to exploit the emotional aspects in the storytelling, we must first define several aspects, required by the EEP model, such as the initial mood and the effect of some actions. The overall parameters have already been presented in Section 4.3.5, as our model examples.

Nevertheless, in order for us to understand the characters' emotions after perceiving events that affect their neighbors, it is important to highlight the proposed relationships among all the different characters in the village. Figure 4.9 shows, in an abstract representation, what the scenario designer would want. This representation is based on a semantic qualification of the relationships that the designer could imagine. The relevant aspect for the EEP model is shown between brackets that represent friendship (positive) or enmity (negative). Although there is a descriptive label in the pairwise relationships.It is important to mention that it is not required to define all the possible combinations of relationships among the characters, those not mentioned here are considered neutral. What is important is to establish the relationship between each NPC character and the player character.

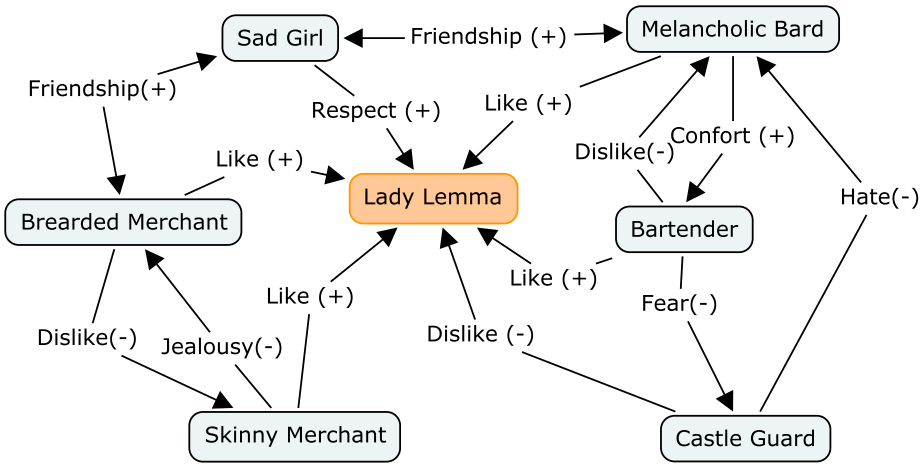


Figure 4.9: Storytelling Scenario: Representation of the Relationships among the Characters in the Village

As an example, in Figure 4.9 there is a FRIENDSHIP relationship between the *Sad Girl* and the *Bearded Merchant*, thus any positive action cast upon one will have a positive influence on the other. For instance, if *Lady Lemma* speaks gently to the girl (which, by itself would help to change the girl's mood from SAD to NEUTRAL), this will influence in the merchant's mood positively (from being NEUTRAL to being HAPPY).

Moods Evolution

For the analysis of the evolution of the moods, we are going to focus our attention on one particular character, the *Bearded Merchant* and also on the direct relationship he

has with the other merchant and with the girl. *Lady Lemma* needs the merchant to become HAPPY so she will be able to purchase from him the rope at a good price (the player character has very limited economic resources). This is one of the possible alternatives to solve one of the steps in the quest. But let's consider two possible alternative sequences of actions:

1. *Lady Lemma* promises the girl to help her find her cat. Additionally, the player interacts with the girl, trying to comfort her from her sorrows, in a gentle and polite way with the girl:
   - The event that produces this action is:
     $$\varepsilon =< Talk, Talk - Polite, \emptyset, Player, SadGirl >$$
   - The corresponding dialogue action casts an emotion of JOY on the girl.
   - But it also produces a HAPPY-FOR emotion on the *Bearded Merchant* (because of their mutual friendship).
2. *Lady Lemma* starts negotiating with the *Skinny Merchant*, but soon after the conversation becomes rude as the merchant's prices are extremely high:
   - This action produces the following event:
     $$\varepsilon =< Talk, Talk - Rude, \emptyset, Player, SkinnyMerchant >$$
   - The corresponding dialogue action casts an emotion of DISTRESS on the *Skinny Merchant*, because this action produces a not desirable consequence (`Talk-Rude`) for him.
   - But it also produces a GLOATING emotion on the *Bearded Merchant* (because he dislikes the other merchant). Due the animosity between the two merchants, the undesirable consequences to a "Foe" character produce an "ill-will" emotion.

These two sequences may lead to the HAPPY mood on the *Bearded Merchant*. To create these sequences, traditionally, we must implement the set of scripts that include the rules of actions and events that might be taken by the player in both cases (and, in any other cases that a creative player could make) to lead the story to the point in which the Merchant is willing to sell the rope at a reasonable price. On the other hand, with the inclusion of the EEP model, we only have to model the relationship and character at a higher level, and set the target moods that trigger the events that make the story to move on.

Likewise, the mood evolution can be controlled by the designer. Only the interesting Mood Space Reference Points are considered as returning values from the EEP, and the bounding of the Mood Vector Space is ensured by definition. Therefore, the possible side effects (in terms of mood variations) of the player's actions are always constrained to the boundaries of the space and the Reference Points described in the profiles of the NPCs. This is necessary in order for the designers to create robust storylines, where the characters are not in any particular mood, but in a constrained set of them.

4.5.4 Evaluation of the Storytelling Scenario

We have selected here an evaluation procedure that is similar to the one presented in the previous section. A group of experienced players interacted with the scenario and, right after the session, they were given a short questionnaire (see Appendix B.4)[7] .

In this case, the evaluation process aimed at two different targets: (1) we wanted to retrieve the different story lines followed by the evaluation players, and (2) we wanted to evaluate how believable the scenario was for the different experts.

In this experiment, the same 16 test players were proposed to play the scenario a maximum of three times, and try to solve the quest right after (given a restricted amount of time, to perform all the actions). From all 16 players, 14 of them successfully finished the quest at least once. About the specific part of the quest of obtaining the rope, 15 of them succeeded. From these successful cases 10 out of the 15 purchased the item from the *Bearded Merchant*, and the rest (5 of them) from the *Skinny Merchant*. In Figure 4.10, we can see that most of the cases in which they obtained the rope from the *Bearded Merchant*, they did it interacting kindly (or at least neutrally) with both the *Sad Girl* and the *Bearded Merchant*. Furthermore, buying the rope from the *Skinny Merchant* is only possible by being extremely polite to him. This actually illustrates both possible paths to solve this part of the quest.

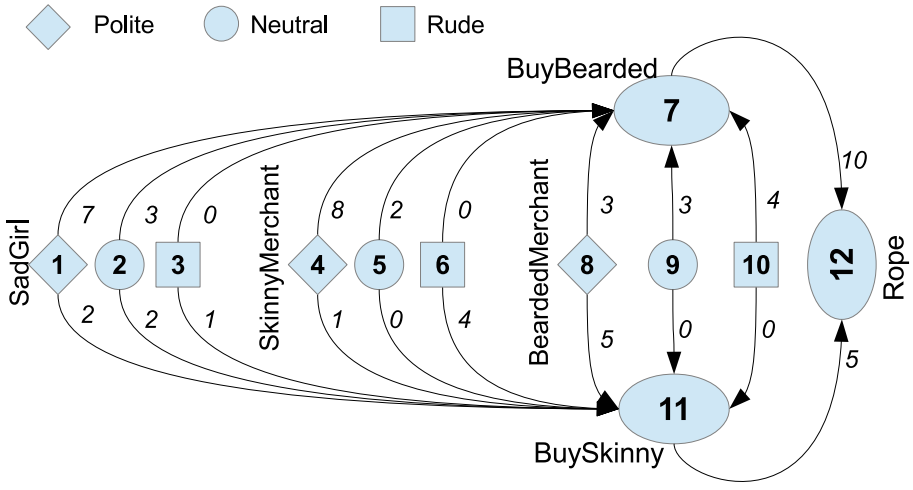


Figure 4.10: Storytelling Scenario: Representation of action dependencies in the evaluation of the scenario. The state numbers represent the different action numbers included in the questionnaire in Appendix B.4.2.

From the second part of the questionnaire, we have obtained the believability scores presented in Table 4.14, as in the case of the Combat Scenario, the believability of the characters is rated from 1 to 5 (no believable to very believable). These figures show that the behavior of the central character, the *Sad Girl*, was perceived as quite reasonable

[7]A video of a played scenario can be shown at http://www.ia.urjc.es/~luispenya/research/eep/

Character	Avg. Believability Score	Std. Dev.
Sad Girl	4.063	±0.9287
Bearded Merchant	3.500	±1.155
Melancholic Bard	3.250	±1.2383
Overall Score	4.438	±0.8921

Table 4.14: Storytelling Scenario: Evaluation results of the questionnaire on the believability of character's behavior according to the information provided to the evaluator. Questions rated from 1 to 5.

(average believability value of 4.063), although, the other two characters seem to be less rational (indeed, the *Melancholic Bard* is sometimes misunderstood, 3.250 of believability with high deviation).

Nevertheless, the overall experience from the scenario was satisfactory (overall believability value of 4.438 ± 0.8921), where the player obtained a rich interaction with the characters, much more colorful and interesting. Indeed, part of the unsuccessful cases where the players had not fulfilled the quest were due to the ill effect caused by a previous action, which directly biased the mood of some key characters against them. We learned here that some of these players realized that emotion-driven scenarios required them to be much more careful with their actions in order to avoid free rampage (as it used to happen in some other scenarios, where the consequences from certain actions were linked neither among them, nor with some other characters that also had a role in the plot).

4.6 Discussion

We are here presenting a new model for the simulation of synthetic emotions in a video game framework. Considering previous models of emotional agents, we created a lightweight model that can work in parallel with the planning model (in our case by selecting different controllers designed with behavior trees, each of them associated with the Moods defined on this scenario). Moreover, this model enriches the inputs of the controller with the mood of the character derived from the emotions.

The requirements proposed in Section 3.2.1 for the Emotional Engine are satisfied as follows:

- **AGCBAR-EmoR1** is observed in the EEP Model in the inclusion of all the Mood Tags, T_i, in the labeling function, $\mathfrak{L}(\mu)$, that establishes a unique tag for each mood state.

- **AGCBAR-EmoR2** and **AGCBAR-EmoR5** are sustained by the MVS that keeps the current mood state of the character along the time, and adds all the emotional

vectors and the decay function (attenuating the past emotions, carrying the mood state toward its initial state) to the character's mood.

- The EB derives the constructions of the consequences of events to a higher element that could have more information about the causal relations of the elements on the environment, leaving the appraisal of the events to the emotional model, thus, we achieve the **AGCBAR-EmoR6** by using the OCC model that presents a structured analysis of events that have a coherent set of emotions.

- The Character Profile includes the information about the personal preferences of the character, meeting the **AGCBAR-EmoR4**, in terms of emotional events, and providing a mechanism to represent the initial and central tendency of the character's mood, $\mathfrak{L}(\mu_0) = T_0$, using the Big Five personality model translated to the MVS, achieving the **AGCBAR-EmoR3**.

- In the **AGCBAR-EmoR7** as we exemplify in the Sections 4.4 and 4.5, the implementation of the EEP model is carried out in a commercial video game engine, with a minor overhead and with the real-time constrains reached; the EEP model is also designed to be implemented in a high parallel architecture (as the GPUs) since it is based on the MVS which is a 3D space of representation.

Finally, the evaluation of the scenario has been presented, according to the criteria of some other researches [45], throughout the subjective quantification by a set of expert gamers that judged that the behavior and reactions of the characters improved compared to those in the original game. Of course, the statistical relevance of our findings is limited, but they do indicate, in terms of emotional "believability", a significant improvement based on the expectation of these environments that the final user had. The two scenarios developed in the evaluation also represent the set of possible applications, in terms of improvement of the playabilty of the next generation of video games, not only with the simulation of emotionally driven behaviors, but also with the inclusion of different ways to develop plots and storylines, so they can achieve higher levels of freedom of action in virtual scenarios.

EEP Model Comparative Analysis

The proposed model can be easily integrated into the AGCBAR Architecture. With this purpose in mind, we developed a stand-alone model of emotions that can interact with the Game Engine: processing the events produced in this engine and using the current mood state as basis for the strategy selection. In our perspective, the mood is the key feature to choose the strategy. Therefore, the strategy is the set of actions chosen to maximize the probabilities to achieve a certain goal. In our approach, the emotion appraisal has a strong cognitive component, and it is the addition of a set of emotions what produces the change in character's general mood state. This change will utterly lead to a change in the strategy of the character.

From this idea and after reviewing the models presented in Section 2.1, we can compare the EEP Model with these models paying special attention to the desired features:

EMA Model

The EMA model presented in Section 2.2.1 has the following drawbacks, if it were to be used in our architecture:

- It needs to infer the future implications of the event observed. This representation of the possible outcomes could be difficult to obtain in terms of a video game because we may need a very large causal chain. This would require much information and time to compute the causal chain correctly. In the EEP Model, we not have to compute a long-term causal chain, because we only want to use the direct causal attribution of a given action and its specific consequence. Thus, the causal inference is processed with the present information needing of no future information or estimation.

- The appraisal and re-appraisal cycle requires past events to be stored in order to present them to the analysis engine. The EEP model only stores, in the MVS, the evaluations of a set of emotions in the format of a transition over the current mood state. Thus, the effects of these transitions are decayed with the attenuation function, $\mathfrak{A}$, making the emotions last as long as they are not attenuated.

- The coping strategies (strategies used to manage the behavior according to the internal and/or external appraised emotions) are proposed to maintain the desirable events and to overturn the undesirable ones. This approach leads to fully emotionally driven characters. The characters controlled by the EMA model select the actions to maximize good emotions. But in terms of a video game, the characters may attend to certain set of strategies predefined (or checked) by the game designers oriented to fulfill certain goals related with the scenario. Thus, the EEP Model delegates the decision of the actions to the Strategies which are oriented to reach the general goals of each scenario, and designed to fulfill them. And the strategy selection is done according to a set of aggregated emotions that takes the character to a certain state.

The EMA model, as we said in the discussion of the previous chapter, is a general agent architecture that is conceived to be used as decision/reaction mechanism for the virtual characters according to the emotions elicited by the environmental events and states. This makes this model difficult to use as part of the AGCBAR Architecture.

ALMA Model

This model is one of the closest to the EEP Model. The different layers of the ALMA model represent the base mood state of a character (derived from his personality), the mood state as a representation of the mid-term emotional state (described in terms

of PAD space) and the emotions produced by a certain action (analyzed with the OCC Model). The main differences with the EEP Model resides are:

- The MVS is an algebraic space that withstands the emotions and the moods in it. The inclusion of the emotions as vectors in this space substitutes the *push-pull phase* of the mood in the ALMA model. This phase has the conception of emotions together with the central mood state as gravitational points.
- The EEP Model is conceived as part of a virtual environment that has implicit relationships among characters, and exploits them, so the emotions produced by the events can be analyzed. The inclusion of the Conceptual Dictionaries facilitates the general definition of appraisal rules (such BE-HIT consequence) that are shared by all the characters in the scenario. So, the rules are easily applied to third-persons events (such, "see a friend BE-HIT by an enemy").

Different Model Features

Marsella, Gratch & Petta [44] compare the EMA and ALMA models, illustrating how their component model framework can highlight conceptual similarities and differences between emotion models. Using the same concepts, we compare, in Table 4.15, the EEP Model with these two other models.

The Person-Environment relationship in the EEP Model is domain dependent because of the necessity of the Conceptual Dictionaries to analyze the event perceived. This modifies the events perceived adapting them to the Appraisal process. The OCC model is used for analyzing the events and for producing a related emotion, but the EEP Model, as the ALMA model, transforms the tagged emotions into their PAD representations. Moreover, the EEP Model uses the MVS to centralize the mood transitions, the emotion aggregation and the management of the decay of emotions.

The actions to be taken are decided out of the EEP Model, to make the strategy creation more controllable by the designer. The cognitive analysis of the consequence of the events is not handled by the EEP Model in order to maintain the computation cost controlled, because the causal attribution and prediction could be computationally expensive.

	EMA	ALMA	*EEP*
Person- Environment Relationship	Domain Independent Decision- BDI + Theoretic Plans	Out of the scope of the model	Domain Dependent Perception - Event Builder Interface
Appraisal-Derivation	Inference over decision-plans	User defined	Events wrapped according to CD
Appraisal-Variables	Lazarus Inspired	OCC Inspired	OCC Inspired
Affect-Derivation	Lazarus-based structural model. It generates discrete emotions and mood states.	OCC-based structural model. It gives "impulses" into core affect.	OCC-based structural model. It projects the emotions into MVS as intensity vectors.
Affect- Intensity	Expected utility model, with threshold model and additive mood derivation.	User defined.	User defined with additive model over the MVS
Affect	Set of appraisal frames, mood with decay (discrete emotion vector)	PAD space representing mood and emotions	PAD space representing mood and emotions with decay
Behavioral-Consequences	The most intense emotions alter behavior display and action selection. Actions are close-loop via domain-independent rules.	Open looped. Mood and emotion alter behavior display and action selection.	Domain-specific actions are handled outside of the model. Mood feedback to decision process.
Cognitive-Consequences	Closed-loop via domain-independent emotion-focused coping strategies that change the BDI items.	Open-looped. Emotions modify intensities of other elicited emotions.	Out of the scope of the model. Consequences are observed in the strategies.

Table 4.15: EEP Model compared with EMA and ALMA. Conceptual components of the models.

Chapter 5

WEREWoLF Model

> I think we all have to fight the werewolf within us somehow.
>
> William Kempe

Traditionally, agent controllers were programmed using ad-hoc implementations, mainly based on finite state machines (FSM). More recently approaches such as Behavior Trees [16] are increasingly used because they are better equipped to deal with more complex behaviors, and because they are relatively easy for human designers to use. Nonetheless, this hard-coded development, when the number of agents and the number of actions per agent are large, makes the design of these controllers time-consuming and error-prone. An alternative mechanism for avoiding the hard-coding of agent controllers is the use of learning mechanisms to develop these controllers. These learning mechanisms can be applied during the development phase of the game, making agents learn the best sequence of actions under different circumstances. Reinforcement learning (RL) is one of the most interesting paradigms in this domain [71]. So far these learning methods have not been very widely used within video games, but this is at least partly due to the unpredictable speed and quality of the learning process. We believe that when learning can be done reliably and quickly, it will be naturally and widely applied in video games development.

State-of-the-art artificial intelligence in video games is taking advantage of these learning techniques that support the game design and its development. Nevertheless, among the different alternatives to carry out this learning process, the most appropriate are those that provide the best trade-off balance between the performance and the learning time.

In this chapter, we present the implementation of a Reinforcement Learning algorithm that is suitable for the video game environment and that fulfills the requirements proposed by the AGCBAR Architecture for the Learning Engine. The proposed algorithm is based on the WoLF algorithm presented in Section 2.3.2. This algorithm is more adequate for the Stochastic Games problems than some other reinforcement learning models. The algorithm developed (called WEREWoLF) is a hybrid reinforcement learning algorithm.

This WEREWoLF algorithm is empirically compared with some other learning algorithms in a context of a one-on-one fighting game.

This chapter is structured as follows: in Section 5.1 we present the integration of the WEREWoLF model in the AGCBAR Architecture as Learning Engine. In Section 5.2 we describe the new algorithm and its components. In Section 5.3, we present a case of study and we evaluate the performance of the algorithm in a stochastic game scenario. Finally, in Section 5.4, we discuss the WEREWoLF model.

5.1 WEREWoLF as Learning Engine in AGCBAR Architecture

The *Automatic Controller Creation* concern of the AGCBAR Architecture is settled in the Learning Engine. This must produce the strategies that are going to be used by the Strategy Selection process to guide the behavior of the characters according to their mood state. In this Chapter, we present the WEREWoLF algorithm as a suitable solution to be implemented in our architecture.

If we observe the Figure 3.7 of the AGCBAR Architecture design, we decompose the Learning Engine into three different elements that correspond to the interfaces interacting with it, see Figure 5.1.

- Environment State Interface to adapt the environment state, so it can be handled by the WEREWoLF algorithm.
- Action Interface to adapt the selected action to the Game Engine.
- Goal Interface to describe the reward and fitness functions that define the current objective of the strategy to create.

Each of them must be implemented for the video game development and must be used in the On-Line and Off-Line phases.

The WEREWoLF algorithm requires a discrete representation of the environment state and a fixed set of actions to be taken. Therefore, the Environment State Interface, Σ, is a discretizer of the environmental state variables.

$$\Sigma : \mathbf{S} \rightarrow \mathcal{S} \subset \mathbb{N}$$
$$\Sigma(\vec{S_i}) \rightarrow s$$

The actions produced by the WEREWoLF algorithm are also a discrete set of actions. These actions are used by the learning process, and they must be translated from the domain used by the WEREWoLF to the domain described in the Actions Dictionary of the AGCBAR Architecture Architectural Dictionaries. This translation is made by the Action Interface,

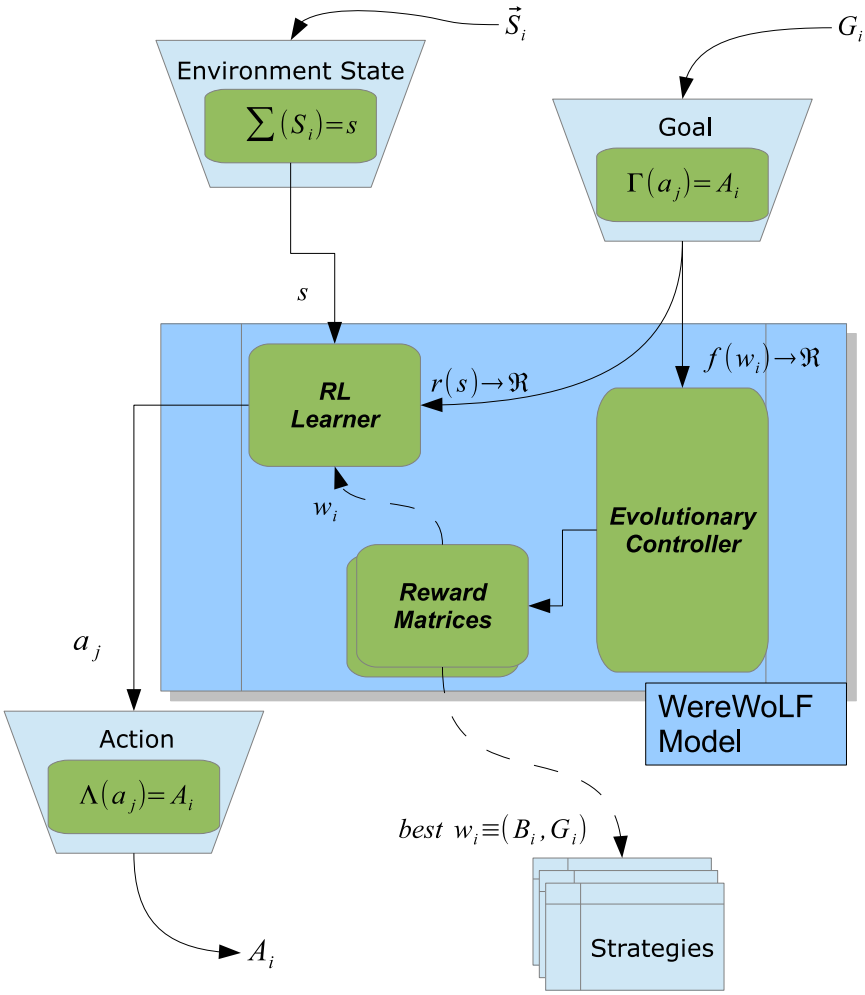


Figure 5.1: WEREWoLF Architecture Analysis. These components are necessary for the integration with the AGCBAR Architecture.

Λ.

$$\Lambda : \mathcal{A} \subset \mathbb{N} \rightarrow \mathbf{A}$$
$$\Lambda(a_i) \rightarrow A_j$$

WEREWoLF exploits the Goals with two purposes: i) to determine the reward function of the reinforcement learning algorithms, and ii) to compute the fitness functions of the Evolutionary Algorithm that controls the population of learners. The reward function evaluates the environment state during the learning of a policy, and the fitness functions determine which of the learners are the best of the population. Thus, the Goal Interface,

Γ, transforms the goal G_i into a pair of functions over the realm of $\Re$.

$$\Gamma(G_i) \rightarrow < r, f >$$
$$r : \mathbf{S} \rightarrow \Re$$
$$f : \mathbf{W} \rightarrow \Re$$

When the learning process of a given goal, G_i, is finished, the best learner individual, w_i, of the population of learners $\mathbf{W}$, becomes the selected strategy, B_i, and it is associated with this goal in the strategy repository. Thus, the *Strategy Repository* updates the association between the strategy and its corresponding goal.

5.2 WEREWoLF Algorithm

WEREWoLF (WoLF Enhanced by Reinforcement and Evolution of the WoLF algorithm) was first introduced in [66] as an hybrid learning mechanism that combines multiple reinforcement learners (originally WoLF learners) using genetic algorithms.

The rationale behind this approach is that: "*there is a performance improvement by combining the learning process of a Q-learning algorithm with the evolutionary operators*". This benefit is due to the synergy derived from (i) the search performed by the learning process, and (ii) the exchange of useful information learned independently by each of the individual by means of genetic recombination.

From this idea, we propose a new hybrid unsupervised machine learning algorithm. This new algorithm will use a population of reinforcement learning instances (e.g. WoLF instances, see Section 2.3.2), learning independently, that are combined after a given number of evaluations in order to obtain a new offspring based on the result of the recombination of learned policies (in the WoLF case, they are both, the Q-values and the probability matrices that represent the learned policy).

5.2.1 WEREWoLF Elements

Thus, in WEREWoLF, schematically represented in the Figure 5.2, we have an Evolutionary Controller wrapping the RL Learner core. The Evolutionary Controller based on an evolutionary algorithm schema.

The different elements of the controller are:

- A population, P_g, of S individuals, $w_1, \ldots, w_s$. Each of them represents the training information of an instance of a Reinforcement Learning Algorithm. This algorithm (the RL Learner core) is, in our case, a variant of the Temporal-Difference learning for the control problem [85] algorithm.

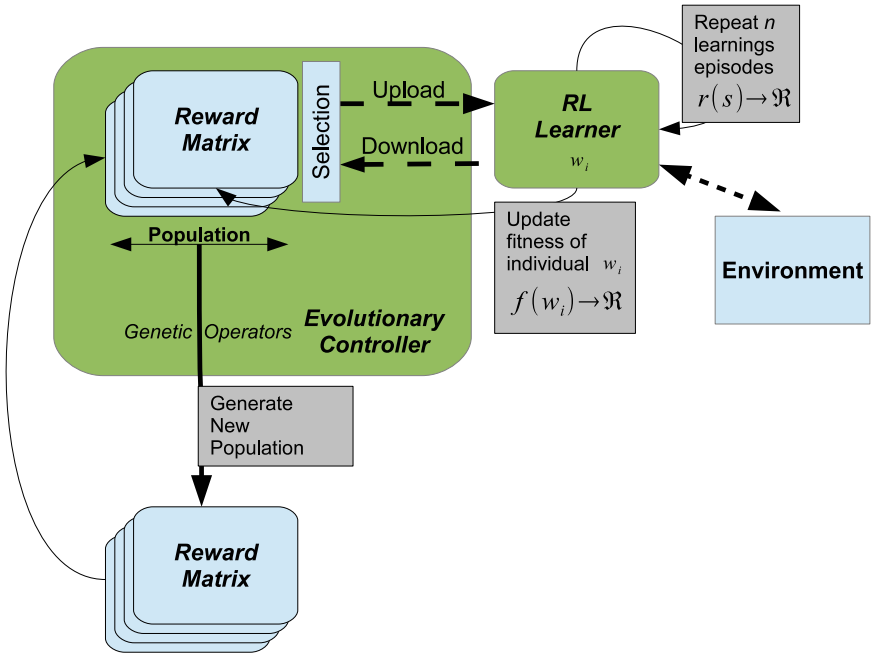


Figure 5.2: WEREWoLF General Schema: The population of the Evolutionary Controller explores the environment independently. The overall performance of the individual is represented by its fitness value. After a number of episodes the population of learners is combined to generate a new one.

- Individuals are encoded by chromosomes which include, for each individual, the action-state expected reward matrix (referred as Reward Matrix). This matrix is the aforementioned training information of the learning algorithms.
- A selection operator. This operator selects the next individual w_i to use its Reward Matrix to train the RL Learner algorithm. This individual tries to achieve the optimal policy that maximizes the outcome according to the reward function $r(s)$. Any changes in the Reward Matrix, resulting from the learning process, are updated in the chromosome code of the individual.
- The learning process is repeated for a number of episodes. After these episodes, each individual computes its fitness, $f(w_i)$, as the average reward obtained for those episodes in which the individual has been selected as the Reward Matrix of the RL Learner.
- When all the individual have updated their fitness values the generation is over.
- A set of genetic operators. These operators generate a new offspring population, P_{g+1} from the current population, P_g. The genetic operators depend on the specific evolutionary algorithm used to construct the Evolutionary Controller.

Therefore, in the WEREWoLF, described in the Algorithm 3, each learner is trained for a fixed number of episodes at each generation. After a complete generation

is finished, the different learners are combined. The new population is obtained by the combination of the individuals. This combination process merges the Reward Matrices to produce a new offspring of individuals. The new individuals in the new population continue learning, using the base RL Learner based on the Reward Matrix provided by each individual.

Algorithm 3: WEREWoLF Algorithm

Initialization: **begin**
  Let P_0 be an initial population of several Reward Matrices of fixed size S
  $\forall_i w_i \in P_0$ initialize $res(w_i) \leftarrow 0$ and $cnt(w_i) \leftarrow 0$
  Let $g \leftarrow 0$ be the generation counter
  Let $c \leftarrow 0$ be the episode counter
end
//The evaluation step is repeated every time this algorithm is selected for evaluation
Evaluation Step begin
  $c \leftarrow c + 1$
  Select w_i from the population P_g and upload the Reward Matrix into the RL Learner
  Train the RL Learner until it reaches a final state s_{final} obtaining $rew = r(s_{final})$
  Update for w_i: $res(w_i) \leftarrow res(w_i) + rew$ and $cnt(w_i) \leftarrow cnt(w_i) + 1$
  Download and update w_i with the final Reward Matrix from the RL Learner
  if $c \mod C_{GEN} = 0$, *the execution counter is multiple of the executions per generation* **then**
    $\forall_i w_i \in P_g$ computes individual fitness $f(w_i) \leftarrow \frac{res(w_i)}{cnt(w_i)}$
    Apply genetic operators over P_g and produce the new offspring population of P_{g+1} and $g \leftarrow g + 1$
    $\forall_i w_i \in P_g$ initialize $res(w_i) \leftarrow 0$ and $cnt(w_i) \leftarrow 0$
  end
end

Specific WEREWoLF Implementation

In order to implement the WEREWoLF algorithm, we selected:

- a specific type of reinforcement learning technique for the RL Learner,
- a specific evolutionary algorithm for the Evolutionary Controller.

We implement, as we will see in the Section 5.3, WoLF and SARSA algorithms as alternatives (for the RL Learner) to include in WEREWoLF. The considered alternatives for the Evolutionary Controller are based on the EDA and DE algorithms.

Therefore, the recombination operators of the Evolutionary Controller must work with the Reward Matrices of the individuals so the new individuals obtained are expected to retain and combine the learned strategies of the best individuals from the past population. Hence, the reward function evaluates the states explored by the individual and the fitness function evaluates the overall performance of the individual.

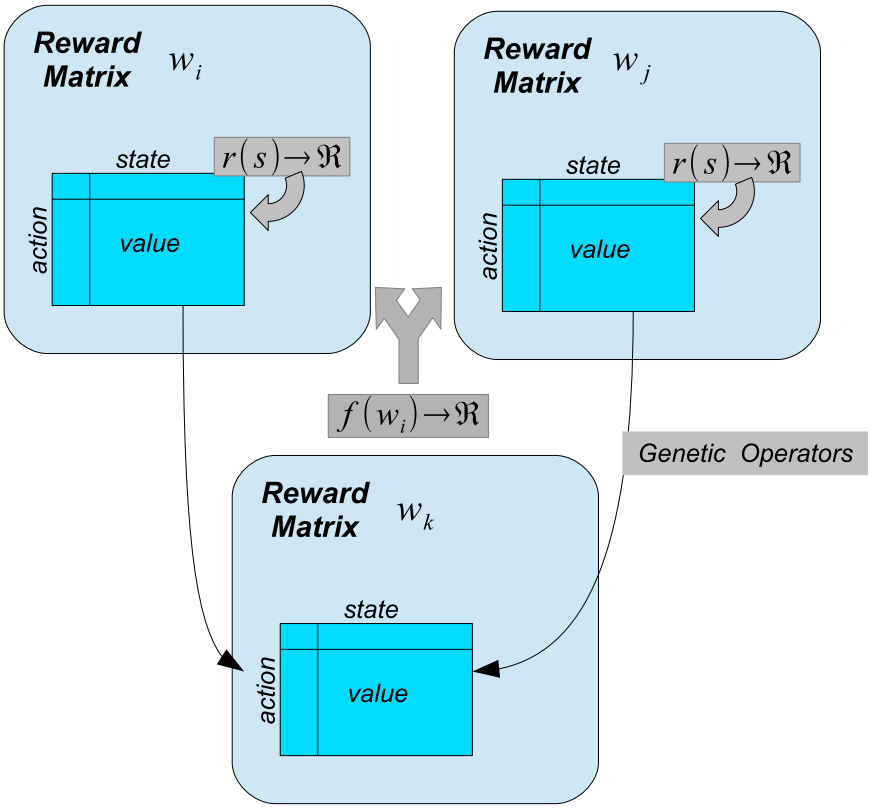


Figure 5.3: WEREWoLF Individual Implementation. The individuals are encoded using their action-state matrix. The genetic operators must combine these matrices to obtain new individuals.

5.3 Evaluating WEREWoLF Performance

The WEREWoLF, as we said previously, is originally based on the Q-Learning variant algorithm WoLF. This algorithm was proved to be a good solution for the Stochastic Game problems. Thus, we use it as one of the reinforcement learning algorithms included in WEREWoLF algorithm, together with the more generalist SARSA approach. In this chapter we validate this assumption and also provide a performance evaluation of the WEREWoLF algorithm over other possible solutions (standalone reinforcement learning). Moreover, we provide a possible implementation of the Evolutionary Controller trying to find a good combination of a evolutionary algorithm and a reinforcement learning.

5.3.1 Experimental Framework

The framework used for the evaluation is based on the vBattle Framework [63], which is a middle-scale-battle video game. In this chapter and for this experiment, we used a simplified version restricted to the one-on-one melee combat engagement. The framework is fully detailed in the Chapter 6.

The main elements of this simulator are the combatants, which are engaged in a

melee combat. The combat finishes after the death of one of the combatants, or when there is a draw because a certain amount of time passed and none of the combatants dies.

Each of the two combatants is managed by a specific controller, that must decide which action it has to take at each step. The combatants have a set of actions and attributes.

An example combatant is included in the Appendix C. We also present here the main components of a combatant:

- A combatant has two different counters that represent his health (called *HP*) and his energy (called *EP*). The different actions that are carried out by each combatant can produce changes at the counters of either or both combatants.

- Each of the combatants has a set of actions that he can execute. The actions are classified into the following categories: `REST`, `ATTACK` and `DEFENSE` actions. The combatant has different detailed actions that belong to these categories, in order to produce interesting asymmetric control strategies, which depend on the combatant being controlled and also on the opponent combatant.

- Each action has two basic counters: The Action Points (*AP*) counter, which represents the amount of time taken from the declaration of the action to the execution or end of the action; and the Exhaustion Points (*EP*) counter, which is the amount of energy consumed (restored in the case of the `REST` action) by the action. This energy consumption/restoration is updated when the action concludes.

- All the `ATTACK` actions have a probability to hit the target, and the description of the basic damage that they produce in the case of hitting such target. The damages are computed with three counters: The Health Damage (*HD*) counter, the Exhaustion Damage (*ED*) counter and the Stun Damage (*SD*) counter. The *HD* and *ED* are subtracted from the respective counters of the target when the attack is successful. The *SD* is the amount of time during which the target is inactive, it also produce the loosing of the target's declared action.

- All the `DEFENSE` actions also have a probability of blocking the attacks. The defenses are active and ready to protect the combatant from the moment they are declared, they also continue protecting the combatant as along as they are active (while the defense action is not finished). If, while the defense is active, an attack is successfully blocked, the amount of *HD* and *ED* damage produced by such attack is reduced by a factor called Reduction (*Red*). There are some defenses (marked as *UnStun*) that can also block the *SD* damage.

The general description of the game engine is explained in the Algorithm 4, in the Appendix C. The main features of this game are: (1) the actions can be declared, in many cases, simultaneously by the two combatants, (2) there is a probability of success/failure in the actions, and (3) the order of execution of the actions can be altered, because there

are certain attacks (those that can inflict stun damage) that can delay the execution of the target's actions. These features make this game an interesting one to study because it captures many of the most important underlying features of fighting games.

5.3.2 Experimental Setup

The experimental setup is based on the contest between two combatants, the first one using a hand-coded static strategy and, the second one using learning to adapt his own strategy over time. For this experiment, the combatants' characteristics are the same for both them with the purpose of being fair and also of making the result dependent just on the strategy and not on the combatant capabilities; the profile used is the one included in Appendix C.

Twelve (12) different controllers, with different rules or mechanisms, are built for the static strategies:

- A random controller, that randomly selects actions from the nine possible ones (`E_RAND`).
- A rule-based engine, with mixed strategies that tries to exploit the time between the declaration and the execution of an enemy action, and selects which fast counter actions should be performed before the opponent finishes his declared action (`E_SMART`).
- Ten controllers based on the Behavior Trees [16, 61], shown in Figures 5.4, 5.5 and 5.6.

Meanwhile, in order to train the different "learning controllers" six proposed alternatives have to be considered:

- A SARSA controller with parameters $\epsilon = 0.1, \gamma = 1.0$.
- A WoLF controller with parameters $\epsilon = 0.1, \alpha = 0.4, \gamma = 0.4, \sigma_l = 0.6$ and $\sigma_w = 0.2$.
- A WEREWoLF controller with a DE evolutionary strategy and the same parameters as the aforementioned WoLF's.
- A WEREWoLF controller with an EDA evolutionary strategy and the same parameters as the aforementioned WoLF's.
- A WERESARSA controller with a DE evolutionary strategy and the same parameters as the aforementioned SARSA's.
- A WERESARSA controller with an EDA evolutionary strategy and the same parameters as the aforementioned SARSA's.

All the "WERE controllers" (WEREWoLF and WERESARSA in their EDA and DE combinations) use a fixed size population of 10 individuals. This individuals are combined to produce the new populations using the two different schemes: DE and EDA.

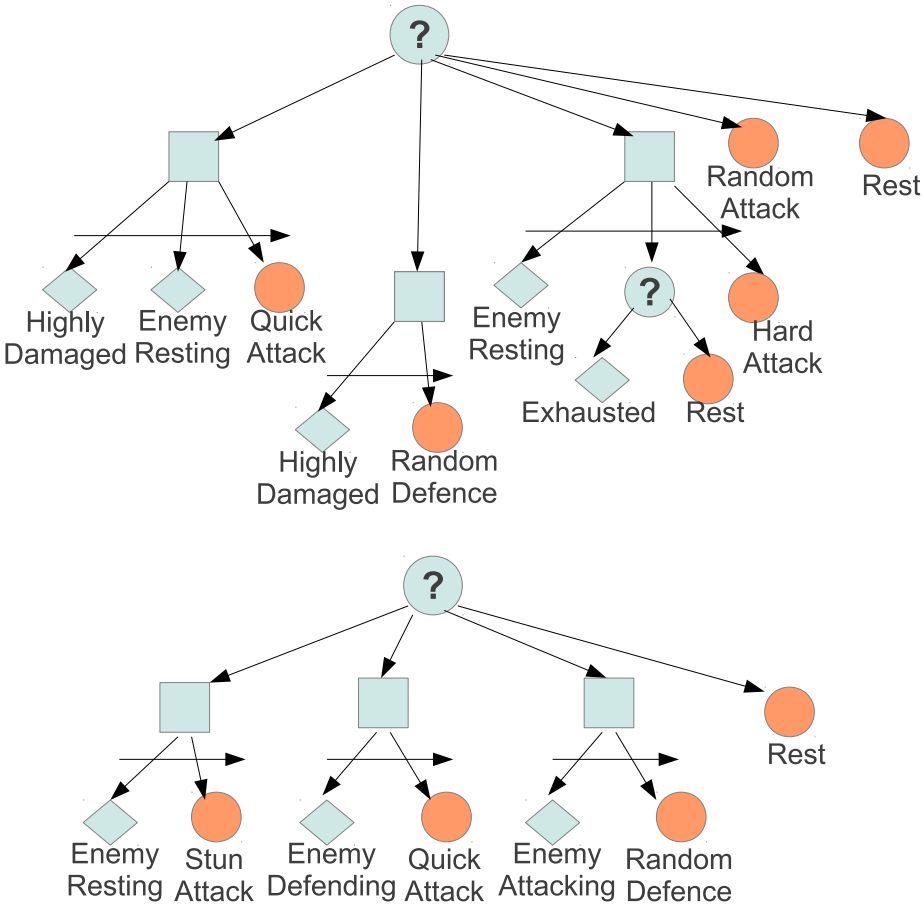


Figure 5.4: WEREWoLF Evaluation: Behavior Trees I. Top: E_BT_OFF Down: E_BT_DEF

Fitness evaluation is the average reward along 20 episodes. DE-specific parameters are: DE/rand/1/bin model, CR=0.5, F=0.5. EDA-specific parameters are: UMDAg model with elitism.

For all of the controllers, the environment is represented by the following state variables:

- The declared enemy action (a value in the interval $[0, 8]$).
- The segment of the action when the enemy action will be executed. These values are discretized into a set of bins depending on the APs required by the different actions the learning controller can do. Thus, each of the bins represents one or more possible actions a controller can perform before the opponent's action is completed. The values of this variable depend on the combatant's profile, using the one depicted in the Appendix the range is $[0, 6] = \{[0 - 6APs], [7 - 10APs], [11 - 12APs], [13 - 15APs], 16APs, [17 - 20APs], 21 + APs\}$
- The percentage of the remaining HPs of the learning combatant is discretized into

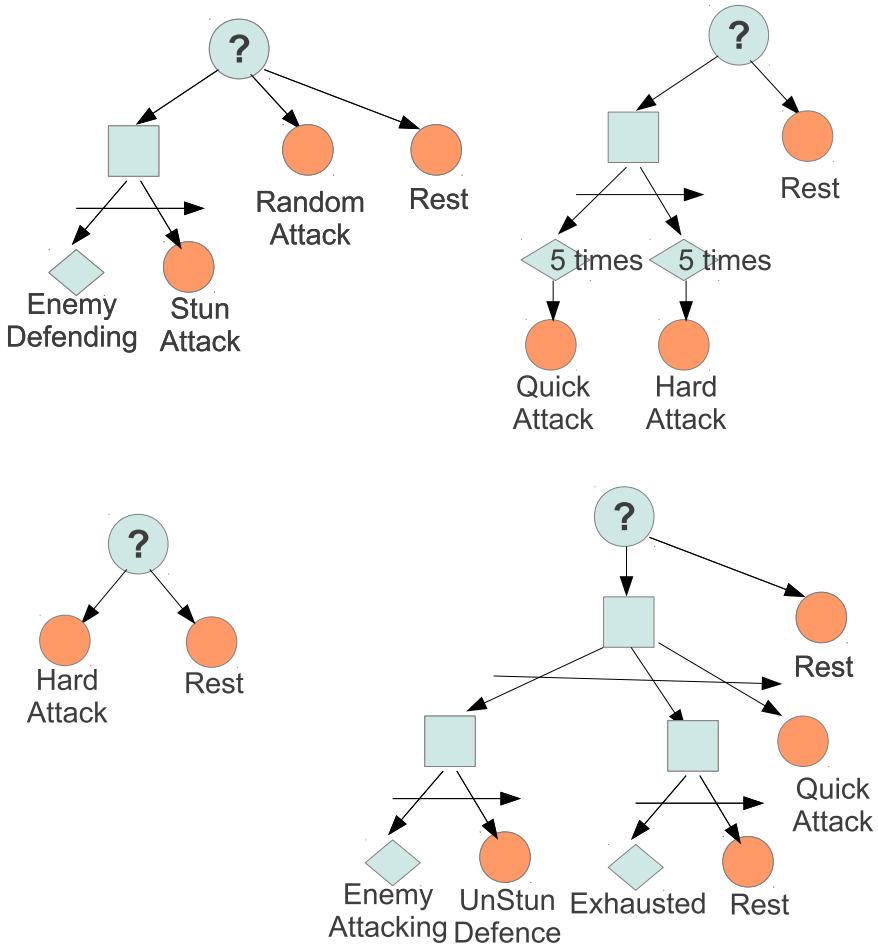


Figure 5.5: WEREWoLF Evaluation: Behavior Trees II. Top to Down, Left to Right: E_BT_ALL, E_BT_COMBO, E_BT_HARD and E_BT_COWARD

four quartiles. The values are in $[0, 3]$

- The relative difference of HPs between the two combatants. If the static combatant has less HPs than the learning combatant the variable is set to 0, otherwise it is 1.

The possible states result from the cross product of these variables (504 possible states) plus four additional states:

- One final state where the combatant controlled by the learner is dead.
- One final state where the combatant controlled by the static strategy is dead.
- One state for the case where learning combatant is so exhausted that he can not take any other action.
- One state applicable in the case the combatant with the static strategy is so exhausted that he can not perform any other action.

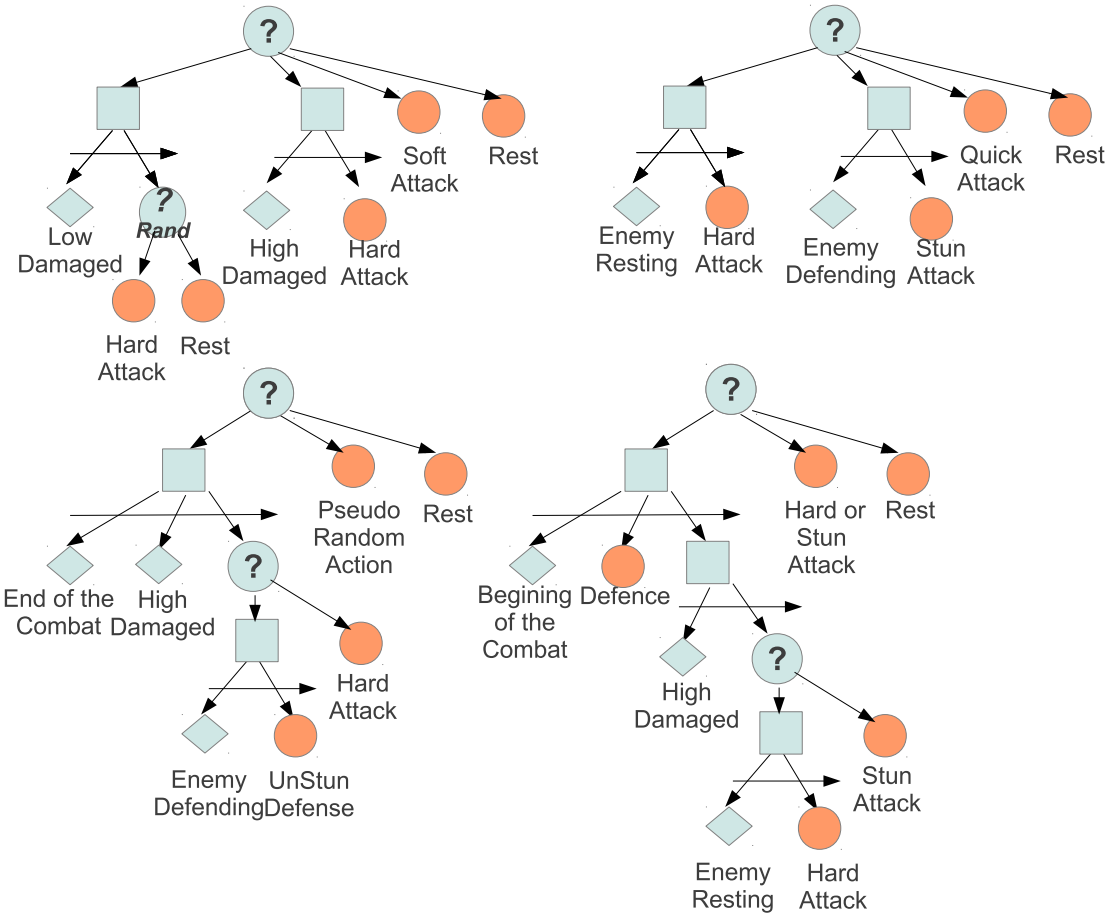


Figure 5.6: WEREWoLF Evaluation: Behavior Trees III. Top to Down, Left to Right: E_BT_ENERGY, E_BT_STUN, E_BT_FINAL and E_BT_TIMER

The complete representation of the environment produces a set of 508 different states. This state space representation is used by all of the controllers. In addition to that, the action space is fixed by the combatant's profile having 9 different actions.

Each step of the simulation obtains a reward derived by the current environment state. The reward function, $r(s) \rightarrow \Re$, is the same for all of the experiments:

$$r = \alpha\Delta\Delta HP\% + (1-\alpha)\Delta\Delta EP\% + \dot{W}(1-\tau)$$

Where,

$$\begin{aligned} \alpha &= 0.95, \\ \Delta\Delta HP\% &= \Delta MyHP\% - \Delta EnemyHP\%, \\ \Delta MyHP\% &= MyHP\%_t - MyHP\%_{t-1}, \quad \text{Same for EP\%} \end{aligned}$$

$$W = \begin{cases} 200 & if \quad EnemyHP < 0, \\ -200 & if \quad MyHP < 0, \\ 0 & otherwise \end{cases}$$

$$\tau = \frac{APs_t}{APs_{max}}$$

The evaluation of the performance of an individual along a generation is done by the fitness function, $f(w_i) \rightarrow \Re$, in our experiment this function is the aggregated sum of the, k, last rewards of each episode obtained by the combatant, w_i, in the current generation.

$$f(w_i) = \sum_k r(s_f)$$

Each experiment has been repeated 25 times (for each combination of one static and one learning controller). Each evaluation is performed as follows: first, a learning phase (out of 10000, 25000 and 50000 episodes), and finally, an evaluation phase (of 1000 episodes). Each episode is carried out until one of the combatants looses (reaching 0 HPs) or until a fixed time of 10000 APs is reached (resulting as a draw combat). The final comparison among all the learning controllers is based on the percentage of victories in the evaluation phase against each of the static controllers.

5.3.3 Experimental Results

In order for us to provide a proper statistic validation of the results, the distribution of all of the results was first compared with the Friedman test. With that, we detect significant differences among the algorithms. In the case of the 10000 episodes scenario, a value of 28.95 was obtained for the chi-squared statistic, which corresponds to a $p - value$ of $2.37E-05$, a value of 33.57 ($p - value$ of $2.90E-06$) and a value of 34.14 ($p - value$ of $2.23E-06$), for 25000 and 50000 episodes, respectively. In the three studied scenarios, these results confirm the existence of significant differences between the algorithms. According to this test, the algorithms are ranked as shown in Table 5.1, where, the WEREWoLF-DE algorithm obtained the best results of all the three scenarios. In addition to that, the table shows the averaged win ratio for each of the algorithms.

After that, two post-hoc methods (Holm and Hochberg) were used to obtain the adjusted p-values for each comparison between the control algorithm (WEREWoLF-DE) and the remaining algorithms. The *Wilcoxon signed rank* test was also used for comparing the results, adjusting the obtained $p - values$ to take into account the Family-Wise Error Rate (FWER) when conducting multiple comparisons. The results of these tests are reported in Table 5.2. These results prove that there are statistical evidences that allow us to state that the WEREWoLF-DE is significantly better than the remaining algorithms (in all cases, according to the Wilcox test and, in the vast majority of cases, according to Holm and Hochberg procedures).

	Ranking			Avg Win Ratio		
	10k	25k	50k	10k	25k	50k
WEREWoLF-DE	1.42	1.33	1.58	0.35	0.41	0.44
WERESARSA-DE	2.75	2.66	2.50	0.26	0.26	0.27
SARSA	3.42	3.08	2.75	0.23	0.24	0.24
WEREWoLF-EDA	3.83	4.17	4.33	0.15	0.16	0.12
WERESARSA-EDA	4.58	4.92	4.83	0.19	0.17	0.16
WoLF	5.00	4.83	5.00	0.19	0.22	0.21

Table 5.1: WEREWoLF Evaluation: Average Ranking and Average Win Ratio against all controllers (10000, 25000, 50000 episodes)

To summarize the general performance of the controllers we can look at the Figures 5.7 and 5.8, that show the average win ratio against each of the controllers (in the particular case of 50000 episodes). The first figure shows the evaluation of the learning controllers against the less effective static controllers. While the second shows those corresponding to the best performing static controllers.

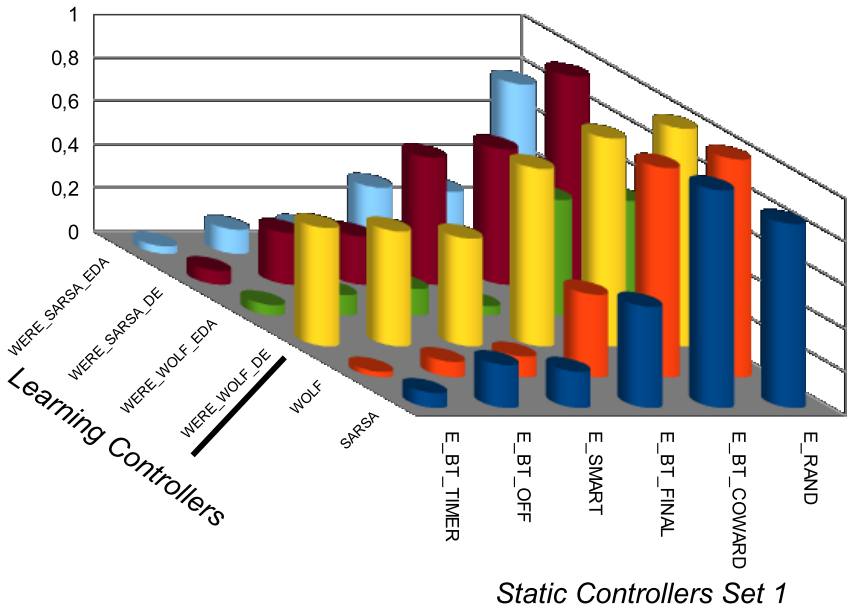


Figure 5.7: WEREWoLF Evaluation: Average win ratio Vs "easy" controllers

In a detailed study of the performance of the algorithms, a particular case behaves differently from the rest of the experiments: The most defensive controller (E_BT_DEF) prefers a draw rather than a loss, tending to consume the time. In this particular case, the SARSA-based controllers reach more victories, but with an average reward lower than

	p-values of the different tests		
WEREWoLF-DE vs.	Holm	Hochberg	Wilcox
10000 episodes			
SARSA	$1.77E-02^{\surd}$	$1.77E-02^{\surd}$	$4.88E-04^{\surd}$
WoLF	$1.35E-05^{\surd}$	$1.35E-05^{\surd}$	$7.32E-04^{\surd}$
WEREWoLF-EDA	$4.67E-03^{\surd}$	$4.67E-03^{\surd}$	$4.88E-04^{\surd}$
WERESARSA-DE	$8.09E-02$	$8.09E-02$	$1.71E-02^{\surd}$
WERESARSA-EDA	$1.35E-04^{\surd}$	$1.35E-04^{\surd}$	$6.10E-03^{\surd}$
Wilcox p-value with FWER: WEREWoLF-DE vs. All			$2.48E-02^{\surd}$
25000 episodes			
SARSA	$4.39E-02^{\surd}$	$4.39E-02^{\surd}$	$4.88E-04^{\surd}$
WoLF	$1.84E-05^{\surd}$	$1.84E-05^{\surd}$	$2.44E-04^{\surd}$
WEREWoLF-EDA	$6.23E-04^{\surd}$	$6.23E-04^{\surd}$	$4.88E-04^{\surd}$
WERESARSA-DE	$8.09E-02$	$8.09E-02$	$1.05E-02^{\surd}$
WERESARSA-EDA	$1.35E-05^{\surd}$	$1.35E-05^{\surd}$	$2.44E-03^{\surd}$
Wilcox p-value with FWER: WEREWoLF-DE vs. All			$1.41E-02^{\surd}$
50000 episodes			
SARSA	$2.53E-01$	$2.30E-01$	$1.22E-03^{\surd}$
WoLF	$3.85E-05^{\surd}$	$3.85E-05^{\surd}$	$1.71E-03^{\surd}$
WEREWoLF-EDA	$9.52E-04^{\surd}$	$9.52E-04^{\surd}$	$4.88E-04^{\surd}$
WERESARSA-DE	$2.53E-01$	$2.30E-01$	$1.34E-02^{\surd}$
WERESARSA-EDA	$8.35E-05^{\surd}$	$8.35E-05^{\surd}$	$3.42E-03^{\surd}$
Wilcox p-value with FWER: WEREWoLF-DE vs. All			$2.02E-02^{\surd}$

$\surd$ means that there are statistical differences with significance level $\alpha = 0.05$

Table 5.2: WEREWoLF Evaluation: Statistical validation 10000, 25000 and 50000 episodes (WEREWoLF-DE is the control algorithm)

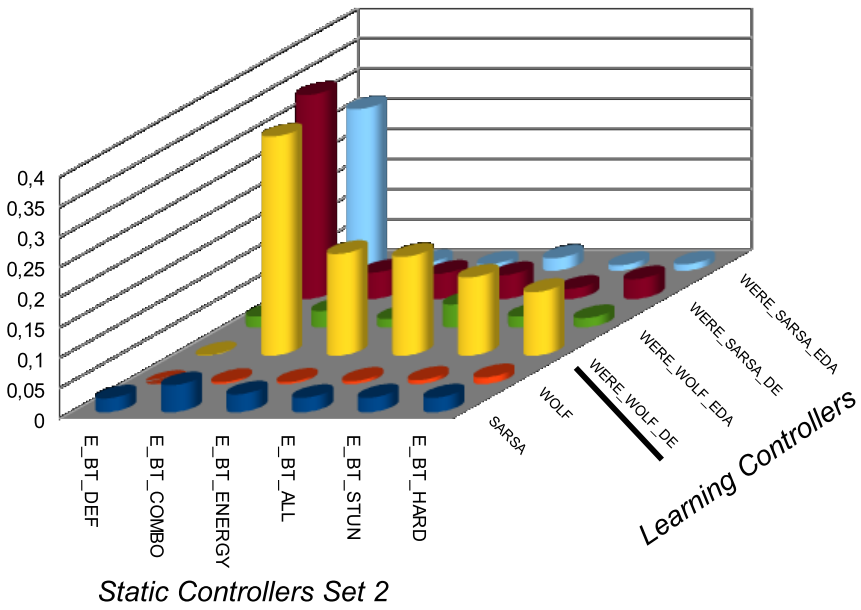


Figure 5.8: WEREWoLF Evaluation: Average win ratio Vs "hard" controllers

the one obtained by the WoLF-based controllers (see Table 5.3). These results also show that, in those problems where the reward function punishes the wrong steps, the WoLF algorithm tends to carry out a more cautious exploration of the environment, becoming a more conservative controller than the SARSA algorithm. In this case, and according to the selected evaluation criteria, we can consider this particular behavior as positive.

	Avg Rw	StdDev Rw	Win Ratio
WoLF	-2.8	22.99	5E-05
WEREWoLF-DE	-4.6	29.27	3E-04
SARSA	-162.6	58.9	0.02
WEREWoLF-EDA	-87.3	100.19	0.02
WERESARSA-EDA	-59.76	153.89	0.26
WERESARSA-DE	-12.19	151.46	0.34

Table 5.3: WEREWoLF Evaluation: Average Reward with 50000 episodes against `E_BT_DEF`

Another interesting study is the evolution of the fitness value. Individuals obtain this fitness value as the average of their performance on each of the episodes they consume during a given generation. Their performance is the sum of all the rewards granted by all their actions in that episode. Although the evolution is not continuously incremental due to the randomness of the problem (actions have only a random chance of success), WEREWoLF-DE shows an incremental trend along the different generations (for instance,

the results shown in Figures 5.9 and 5.10).

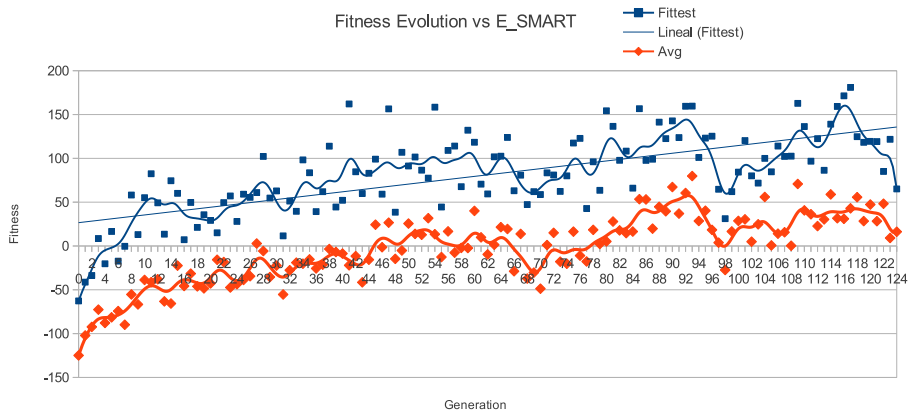


Figure 5.9: WEREWoLF Evaluation: Fitness evolution vs E_SMART (50000 episodes)

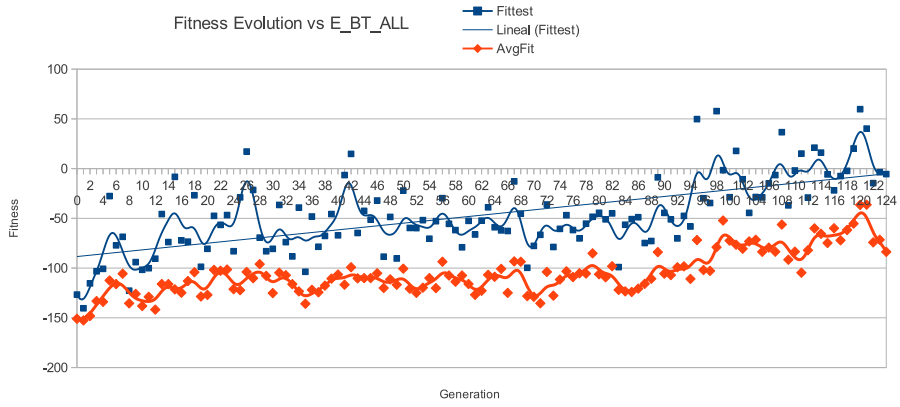


Figure 5.10: WEREWoLF Evaluation: Fitness evolution vs E_BT_ALL (50000 episodes)

Finally, we can say that it is obvious that the DE search exploits much better the mechanism for combining multiple learners in this evolutionary framework. The results are consistent for both, the WoLF and the SARSA controller cores through multiple experiments (10000 episodes to 50000 episodes). The poor performance of the EDA variants can be explained by the reduced population (compared with the size of the chromosome). This was also a reason why other state-of-the-art optimization techniques, such as the CMAES, were not taken into account. The DE proves to be a powerful optimization alternative for high-dimensionality problems.

5.4 Discussion

We implemented and tested two hybrid learning algorithms that combine evolutionary algorithms with reinforcement learning. We also compared the controllers produced using these algorithms with several other controllers, and analyzed the results, using standard statistical tests. The results show the improved performance of the reinforcement learning algorithms when combined with evolutionary techniques. This experimentation was carried out in environments with large state spaces and with a set of complex actions where the "WERE-Hybrid" algorithms outperformed the exploration of the standard reinforcement learning mechanisms.

Despite the improvement of the learning algorithm, this work highlights some interesting facts. The representation of the state space and the construction of the reward function is, like in any RL problem, complex and critical for the success of this experiment. The results show that the application of the RL to this kind of video game scenario is highly dependent on the environment variables used for the description of the state and the reward function, that must provide enough information to guide the learning process.

Therefore, the AGCBAR Architecture provides a basic structure for the definition of the actions and states, through the definition of the interfaces. This structure enables the integration of the WEREWoLF algorithm with the Game Engines. In the Environment State Interface, we must specify the information of the environment state, while trying to maintain the number of different states controlled. We must also provide, in the Action Interface, the actions to build the strategies. In this case, we can abstract the action definitions, in order to keep the number of actions as small as possible. As we will see in the next chapter, the inclusion of the WEREWoLF algorithm could be made thanks to the leveled division strategy, that helped us keep the learning problems in tractable size.

Chapter 6

vBattle Experimentation Framework

> The general who wins the battle makes many calculations in his temple before the battle is fought. The general who loses makes but few calculations beforehand.
>
> Sun Tzu

In this final chapter, we will illustrate the AGCBAR Architecture integration into a simplified, but yet complete, video game. Our main objective is to present a complete experimental setup, that applies the different AGCBAR Architecture features, as a proof-of-concept of the benefits of the contributions derived from this thesis.

Because of the limitations (due to license or technical problems) for the use of many commercial frameworks, we decided to create our own simulation framework, named vBattle (described in Section 6.1). vBattle [63] is a tactical combat video game, similar to some commercial titles, like Fallen Enchantress[TM1] or XCom Enemy Unknown[TM2].

The main features of our framework are:

- The game engine, the rules and the complexity of the environment are similar to the ones of some commercial video games.
- The implementation is open for research purposes, leaving the optimal performance aside, and enhancing the modularity of the different components.
- The interfaces among the modules are implemented according to the specifications extracted from the AGCBAR Architecture.

[1] http://www.elementalgame.com/fallen-enchantress

[2] http://www.xcom.com/enemyunknown/

For our experimental scenario, the Engines specified in the AGCBAR Architecture are implemented by the different models presented through the previous chapters. For instance, the Emotional Engine used is the EEP Model described in Chapter 4, and the WEREWoLF Algorithm presented in Chapter 5 implements the Learning Engine, along the same line, the Game Engine is the set of Core Rules of the vBattle.

This chapter is organized as follows: First, in Section 6.1, we will present the general design of the vBattle framework. In Section 6.1.6, we will describe the implementation of the vBattle and how we adapted it to the AGCBAR Architecture. In Section 6.2, we will show our experimental results. Finally, in Section 6.3, we will discuss the contribution of this framework and the application of the AGCBAR Architecture to the development of complex video game scenarios.

6.1 vBattle Game Design

The vBattle game core is designed following the specifications of the game design described by Brathwaite & Schreiber [12]. We will explain here the set of elements, rules and features that composes the complete game engine. After the formal definition of the rules of the game, we will describe the different components that will be implemented and the relationships among them. In order to clarify some of the concepts,we added in the Appendix A.1 a compendium of commonly-used terms that are related to the depiction of the game.

6.1.1 Game Concept

The vBattle is conceived as a tactical combat game, where two (or more) factions composed by several sets of warriors fight on a battlefield. The objectives of the factions (groups of warriors) in the game are related with the specific scenario goals. The goals of the scenario go from killing all the enemies on the battlefield, to reach and keep a determined position on the game map for a certain period of time. In order to achieve these goals, each faction has to plan where to move its warriors, what is the appropriate action to perform, and how to use the favorable characteristics of the terrain.

The factions are heterogeneous, composed by different types of warriors. Each of these warriors has his own characteristic, actions and attributes. The scenario describes the composition of the factions and the terrain that conforms the battlefield, as well as the goals that each faction should achieve.

6.1.2 Game Elements: Players, Avatars and Game Bits

The vBattle game engine is divided into the following elements:

Element	Definition
Battlefield	It is made by an hexagonal grid board that shows a certain terrain elements (with different shape, size and properties).
Two or more Factions	They are groups of allied Combatants that are controlled by a Faction Leader.
Faction Leader (FL)	He is the commanding intelligence in charge of an entire Faction. High-level orders are issued by the Faction Leader.
The Combatants	They represent warriors that act in the game board to achieve the objective of their Faction. They are the final actors that interacts with the environment.
The Commanders	They are the Combatants in charge of forwarding commanding orders issued by the Faction Leader to the rest of the Combatants. Each Commander's and scenario characteristics can modify the transmission of the FL orders.
The Events	They are the tokens that are distributed along the time sequence of the game. They identify the precise moments when a previously declared action is executed.
The Event Sequence	It is the ordered sequence of Events associated to the current scenario.
The Current Instant Counter	It identifies the current time step of the game. It is used to manage the schedule of the events in the Event Sequence.

Table 6.1: vBattle Basic Elements

The Battlefield

The Battlefield is a representation of the terrain. It is divided by a regular hexagonal grid. Each hexagonal Cell of the board contains different elements, see Figure 6.1. The cells have the following set of properties:

Element	Definition
Identifier	The cells are described by a unique identifier in the battlefield.
Movement Penalty	It represents the relative difficulty to transit through this cell. This property is derived directly from the type of terrain that it represents. This penalty modifies the Base Movement Rate of the Combatant. See the Movement Rules in Section 6.1.3.

Continued on next page

Table 6.2 – *Continued from previous page*

Element	**Definition**
Fixed Objects	This kind of Objects can not be moved from the cell where they are set (i.e, a tree or rock). These objects have the physical properties derived from their dimensions, offering a defensive covering, according to their size and location. Some objects forbid the movement in that hexagonal cell, and some others makes the movement through the cell more difficult.
Mobile Objects	These objects can be moved from one cell to another. They may also provide defensive covering in the same way the fixed objects do.
Logical Objects	These objects represent the special elements that are associated to each particular cell. For instance, the cells can include a logical object of "Key Location". Key Locations are special positions on the battlefield. They represent important conditions in particular game scenarios.

Table 6.2: vBattle Battlefield Elements

There are two special type of regions (sets of cells) in the battlefield: the Escape Zones and the Deploy Regions. The Escape Zones are cells, usually located right at the border of the battlefield. When a Combatant reaches one of these cells he is removed from the game and he is considered as "Fled". The Deploy Regions are sets of cells (usually arranged in clusters) where the Combatants start the game. There are different regions associated to each Faction.

The Factions

The Factions are identified by different names and colors. Each Faction is composed by a set of Combatants. The objectives of the scenario are described in terms of Factions, i.e. each Faction has its own goals in a scenario. Each Faction also has the information of how many Commanders it has and how many it could eventually end up with. All the Factions are controlled by a Faction Leader (FL) that regulates all the Combatants in the Faction.

The Faction Leaders

The Faction Leader(FL) is a high-level intelligence that commands an entire Faction. The FL can perform a set of different actions that are combined to obtain a variety of plans or strategies, in order to achieve the current goal of the Faction. The FL actions are listed in Section 6.1.3. These actions, issued by the FL, are intended to be assigned to

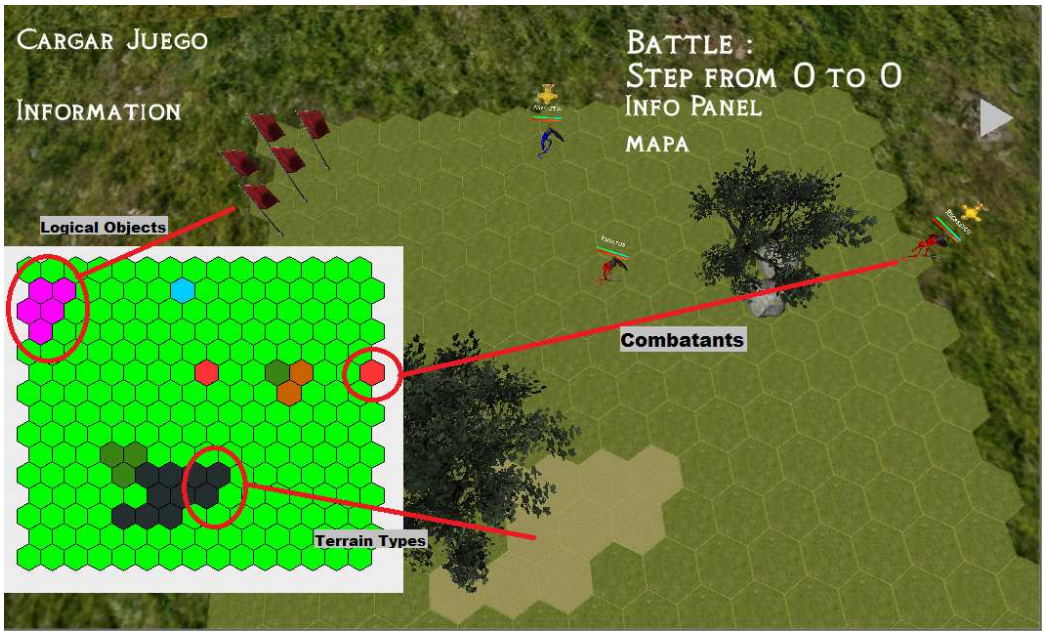


Figure 6.1: vBattle battlefield elements. The battlefield represents a terrain with certain objects, some fixed objects (trees, rocks, ...) and some logic objects (flee zones, ...)

a specific Combatant that will execute the most appropriate action, according to his actual objective, state and behavior.

The Combatants

The Combatants are the warriors that are involved on the battle. They are represented by an avatar on the battlefield. The Combatants' characteristics are described in Table 6.3, and their actions in Section 6.1.3.

Element	**Definition**
Name/Identifier	It is the unique identifier of each combatant on the battlefield.
Hit Points	This represents how much life a combatant has left before dying.
Fatigue Points	They are the vigor of the warrior to perform actions. These points are consumed and recovered during the battle according to the actions carried out by the Combatant.
Armor	It represents the resistance to the enemies' blows. It decreases the amount of damage caused by the enemies' hits.
Base Movement Rate (BMR)	It is the default time needed by a Combatant to move from one cell to another. This value is modified by the movement penalty of the cell.

Continued on next page

Table 6.3 – *Continued from previous page*

Element	**Definition**
Leadership	It is the skill to forward the FL's orders, when this Combatant acts as a Commander.
Discipline	It indicates the likelihood of accomplishing the orders commanded by the Faction Leader and forwarded by the closest Commander.
Block Skill	It is the Defensive Bonus that a Combatant has when he tries to block an incoming attack.
Dodge Skill	It is the Defensive Bonus that a Combatant has when he tries to dodge an incoming attack.
Attack Skill	It represents the Base Attack Bonus of a Combatant. It could be modified by the type of attack selected.
Melee Attacks	They are the attacks that a Combatant can carry out when he is engaged in a hand to hand combat. Each of the Melee Attacks has the following information: • Identifier: Name of the attack. • Execution Time: The amount of time spent from the instant when the action is declared, to the instant when the action is executed. • Attack Modifier: It can be either a positive or a negative change of the Attack Skill. • Fatigue: It refers to the amount of Fatigue Points that this attack consumes during its execution. • Base Damage: It is the range of damage that this attack can produce. • Extra Damage: It is the range of damage that it is added to the Base Damage for every Damage Category of a hit.
Projectile Attacks	They are the attacks that a Combatant can perform from the distance. Each of the Projectile Attacks has the following information: • Identifier: Name of the attack. • Execution Time: The amount of time spent from the instant when the action is declared, to the instant when the action is executed. • Attack Modifier: It can be either a positive or a negative change of the Attack Skill. • Fatigue: It refers to the amount of Fatigue Points that this attack consumes during its execution.

Continued on next page

Table 6.3 – *Continued from previous page*

Element	Definition
	• Range: It is the effective range of this attack represented in number of cells. This attack can not be used to hit enemies farther than this number cells. • Base Damage: It is the range of damage that this attack can produce. • Extra Damage:It is the range of damage that it is added to the Base Damage for every Damage Category of a hit.

Table 6.3: vBattle Combatant's Characteristics. Each Combatant has a set of Melee and Projectile Attacks.

The Commanders

The Commanders are special Combatants that are in charge of a group of other Combatants from a given Faction. The FL selects which Combatants will be the Commanders. Each scenario defines how many Commander every Faction can have at a given time.

When the FL issues an order to a Combatant the nearest Commander is used as a beacon to forward the order,

Commanders behave just as any regular Combatant. The only difference happens when the FL issues a new command. The Leadership of the Commander is one of the parameters for computing the commanding influence of the order.

Event Sequence

The events are stored in an ordered queue. As the game goes on, it processes these events in order. At each instant, the game engine extracts and processes the next event from this queue. This structure supports the Discrete Event Simulation which is part of the game dynamics.

A specific instant in the Event Sequence can have more than one event. In this case, the order of execution of the events is selected randomly.

Current Instant Counter

The scenarios begin in the instant zero (0). This value is stored in the Current Instant Counter. As the game moves on, this counter increases. Each time that an action is declared and scheduled, the value of this counter is used to calculate the instant when the action is executed.

6.1.3 Game Mechanisms

We call mechanisms the internal procedures that rule the dynamics, effects and results of the game actions and progression.

Set up Mechanism

First, the scenario is configured according to the selected terrain, victory conditions, elements involved and their properties.

Initially, the different Factions are defined as sets of Combatants, each of these Combatants embodies a warrior with different characteristics, actions and personality. The Combatants of the different Factions are placed on the battlefield at specific initial locations given by the scenario (Deploy Regions). The Escape Zones are also set, usually on the borders of the battlefield. Finally, if the scenario requires some special zones, such as the Key Places, they will be set at various specific cells.

Game Dynamics

The game is ruled by a Discrete Event Simulation process that consumes the Event Sequence. This Event Sequence queues the pending actions when they are declared by the Combatants, or when they are prompted by a random event, see Figure 6.2. The events represent the instants when actions are expected to be executed.

The game dynamics also uses the Current Instant Counter, which represents the time-step corresponding to the last event processed by the game engine.

The declaration-action sequence is structured as follows:

1. When a Combatant has no action declared: He declares the next action to do.
2. This action is queued in the Event Sequence with the future time stamp, obtained by adding the Current Instant Counter to the Execution Time associated with the declared action.
3. The Current Instant Counter moves the time stamp of the first event in the Event Sequence forward.
4. An Event is executed and removed fomr the Event Sequence when the Current Instant Counter reaches the time stamp referred by that particular Event.

The following are the different types of Events that are included in the Event Sequence:

- **Movement**: Along the event sequence the movement of the units is done in parallel to the action declaration and execution. Eventually, the Combatants can move to any adjacent cell from their current position. The amount of time required for

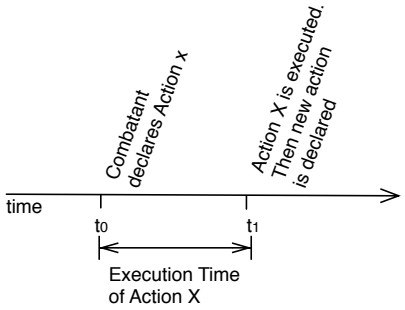


Figure 6.2: vBattle timeline of the event sequence. The actions require certain execution time. The execution time is added to the current timestamp when the action is declared, in order for the game engine to compute the time stamp when the event is executed and enqueued into the Event Sequence.

the Combatants to move from one cell to any of its neighbors depends on the BMR (Base Movement Rate) of each Combatant and the Movement Penalties of the terrain. Movement actions are then a series of independent events queued in the Event Sequence, according to each Combatant's Base Movement Rate. Therefore, each Combatant could have two queued events in the Event Sequence, one related to his next Movement action and another related to his current declared action.

- **Run Action**: A Combatant can declare the Run Action as his regular action. Once this action has been declared, the Combatant cannot perform any other action for a certain amount of time (usually 10 time units). When the Run Action is active the BMR of the Combatant is halved. Also, while a Combatant is running he is spending a certain amount of Fatigue Points.

- **Attack Resolution**: The Melee Attacks and The Projectile Attacks are resolved as follows:

  1. The attacker declares the attack to prepare. The execution of this attack is delayed, depending on the execution time required by the action.
  2. When the action is executed, the attack takes place. The attacker declares the enemy Combatant who is the target of his attack. If there is no target, the attacker losses this action.
  3. The attacker loses the amount of Fatigue Point that this attack consumes.
  4. The defender decides which defense he performs, depending on his current declared action, the defensive action could only be Dodge if the incoming attack is a projectile attack.
  5. The attacker calculates his `AttackRoll` for his attack adding a random value

from 1 to 20 (open ended[3]) to the Attack Bonus[4] of the selected attack.

6. The defender calculates his `DefenseRoll` for his defense adding a random value from 1 to 20 (open ended) to the Defensive Bonus[5] of the selected defense.
7. The `AttackResult` is calculated as: $\texttt{AttackResult} = \texttt{AttackRoll} - \texttt{DefenseRoll}$
   (a) If the $\texttt{AttackResult} <= 0$ then the attack misses its target.
   (b) If the $\texttt{AttackResult} > 0$ then the attack hits its target
      – The Damage Category $\texttt{DamCat} = (\texttt{AttackResult}/10)$ round down.
      – The $\texttt{Damage} = \texttt{BaseAttackDamage} + \sum_{\texttt{DamCat}} \texttt{ExtraAttackDamage}$
      – The $\texttt{Wounds} = \texttt{Damage} - \texttt{Armor}$
      – The target receives `Wounds` counted as the damage on his HitPoints. For every 10 points of `Wounds` over 10, the target loses 1 point of AttackBonus and DefenseBonus. The defender must spend from 1 to 10 more time steps to complete his current declared action (the target is stunned).

- **Assign Commander**: When a Combatant that is currently selected as Commander dies or flees, an *uncoordinated time* begins. During this time, the orders sent to the Combatants no longer come from the dead Commander as a beacon. When this uncoordinated time begins a new *Assign Commander* event is scheduled in order for the Faction Leader to select a new Commander.

Possible Actions

There are two different controllers that decide which different levels of actions: the Faction Leader and the Combatant. Each of these controllers has a different action granularity.

- **Faction Leader's Actions**: The actions available for the Faction Leader (FL) are related to the high level plans. These actions are composed by: a command, a combatant and a target cell:

Element	Definition
Area-Fire	The FL orders a Combatant to fire his projectile weapon to the units placed in a certain location of the map, the area of possible fire attack has a radius of three (3) hexagons.

Continued on next page

[3] If the result is 20 then we add another value from 1 to 20, and so on.

[4] $AttackBonus = AttackSkill + AttackModifier$

[5] $DefenseBonus = [BlockSkill|DodgeSkill] + DefenseModifier$

Table 6.4 – *Continued from previous page*

Action	**Definition**
Fire-At	The FL orders a Combatant to fire to the Combatant placed in the designed location and also to keep firing at him till the target Combatant is dead. The location area is an hexagonal cell that contains an enemy Combatant.
Defend-Position	The Combatant must attack any unit that penetrates a selected area. This area is identified by its central cell. The area is of 3hex/radius.
Take-Position	The Combatant must move toward a selected cell and must attack any unit that is on his way to the designed location. The location has an area of 1hex.
Attack-To	The FL orders to attack a certain Combatant and to keep attacking and hunting him until that target Combatant is dead. The attack can be done just with melee weapons. The location area is an hexagonal cell containing an enemy Combatant.

Table 6.4: vBattle Faction Leader's Actions

The idea behind the FL's actions is to give specific tasks to each of his combatants. The detailed movements or paths to accomplish such tasks are not given by the FL, but decided by each Combatant. The orders commanded by the FL are forwarded by the Commanders. Therefore, the Combatants receive the orders with certain influence based on the Commander's Leadership Skill, and on the inversely proportional distance between the Commander and the Combatant.

Therefore, when a Faction Leader issues orders to his Combatants. The issued Order is received by each Combatant with an Order Strength (OS) equals Commander's Leadership (CL), but modified by the distance between the Commander and the Combatant. The OS is calculated as follows:

$$OS = CL \cdot \frac{\sigma}{distance}\text{, where } \sigma \text{ is a scenario factor}$$

When the Faction Leader commands a Combatant to perform an action, the following condition must be accomplished:

$$D20 + OS + CD > 25$$

Where $D20$ is a random number between 1 and 20 and the CD is the Combatant's Discipline attribute.

- **Combatant's Actions**: The Combatants can carry out three different types of actions:

Action Group	**Definition**
Movement Actions	These actions are performed in parallel with the regular combat actions. The Event Sequence keeps the time stamps that indicates when each one of the Combatants can move, depending on the BMR and the cell Movement Penalties.
Combat Actions	These actions are decided by the Combatant when he has no declared action. The action is carried out after a set amount of time, depending on how long the execution of the chosen action takes.
Defense Actions	These actions must be decided by an agent every time he is the target of an attack. They refer to the type of defense the agent chooses to protect himself against the incoming attack.

Table 6.5: vBattle Combatant's Action Groups

The *Movement Actions* represent the selection of the next hexagonal cell the Combatant moves to. Every combatant has a BMR that indicates the time required to move from one cell to the next.

The *Combat Actions* of the Combatants are the following:

Element	**Definition**
Move-To	It implies the selection of one of the six adjacent hexagonal cells, a Combatant wants to move next.
Attack-{1-n}	It identify the intention to hit an enemy with a melee attack.
Fire-{1-n}	It is a projectile firing against the enemies.
Rest	It refers to the Combatant's waiting period during x amount of time steps, so he can recover some of his FatiguePoints.

Table 6.6: vBattle Combatants' Combat Actions

The *Defensive Actions* are those actions executed by a Combatant when he is the target of an attack. These actions are declared and executed at the same instant the incoming attack takes place. The effect of these actions can delay the execution time of the current action, that was already declared by the attacked Combatant. For example

> *... a Combatant A decides to Attack with the enemy with his sword and his action is scheduled in the Event Sequence at* $t = 35$*. If he is attacked at the instant* $t = 30$ *(by somebody) and A decides to dodge this incoming attack, he will have to delay his attack until* $t = 38$ *(the action is delayed by 3 units later).*

The possible defensive actions to be taken are:

Element	**Definition**
No-Defense	The Combatant will not try to defend himself, which is translated to a huge penalty given to the Dodge Ability. But, it consumes neither Fatigue Points nor time steps.
Soft Block	A less time-consuming defensive action. It applies a penalty to the defender's Block Skill, but does not consume Fatigue Points.
Block	The default defense of the combatant when he is in melee combat.
Full Block	Very time-consuming action but with the bonus of an improved defense.
Dodge	This is a special type of defense that must be carried out after the combatant has declared that he is going to run or fire a projectile.

Table 6.7: vBattle Combatants' Defensive Actions

6.1.4 Goals

The Faction Leaders have the goal of winning the battle according to these possible scenarios:

1. *Kill'em All*: A simple scenario in which a faction wins if all the combatants of the other faction are dead of have fled the battlefield.

2. *Being the best killer*: A scenario in which wins the faction that kills more enemies at a given time.

3. Special Scenarios: These scenarios use the Key Places to define the victory conditions of the game. Some special scenarios considered in this game are:

   - *Take that flag!*: This scenario sets some specials items (Flags) at some particular locations on the map. When a Combatant reaches a Flag he is entitled to hold this flag. After this action, that Combatant must reach a predefined Key Place to deploy the flag. When the Combatant succeeds at carrying the flag to this Key Place, the flag is considered to be safe. If the Combatant that took the Flag is killed, the Flag will lay on the cell where the Combatant died. The game ends when every Flag is safe, or when there is only one Faction with active Combatants.
   - *Hold on!*: In this scenario one Combatant from a given Faction must stay at the selected Key Places for a minimum amount of time, defending this position. The game ends when the allotted time has passed and there is, at least, one

Combatant remaining at the Key Place. If all the Combatants are dead or fled from the battle the Faction loses the battle.

The Combatants have their own goals:

1. *Obey the Orders*: It refers to the Combatant's intention to fulfill the commanding order issued by the FL, and forwarded by the Commander.
2. *Stand and Fight*: The Combatant has no will to obey any other command, but to stand and fight.
3. *Pursuit of Fleeing Enemies*: The Combatant will run after the opponents that are fleeing from the battlefield. If the fleeing unit reaches an Escape Zone this goal is canceled.
4. *Flee and Survive*: The Combatant only wants to survive, and, to achieve that, he runs to any Escape Zone on the map.

6.1.5 Game State and Visible Information

At any instant (see Figure 6.3), the Game State can be described by the following information:

- The Current Instant Counter.
- The current Faction state:
  - The current Commanders assigned.
  - The remaining Combatants alive and their current states (HPs and FPs).
- The current Event Sequence: It refers to the events that are going to be executed in this instant of the Even Sequence.
- The Key Places on the battlefield and their states.

Moreover, the following information is also available for the Faction Leaders and for all the Combatants:

- The information related to any cell on the battlefield.
- The current level of health and fatigue of any Combatant.
- Whether a Combatant is a Commander or not.
- The current action declared by a Combatant.
- The location of any special item (as Flags) on the battlefield.

- The Escape/Flee Zones on the battlefield.
- The Key Places throughout the battlefield.
- The Field-of-View and the coverings of any Combatant.

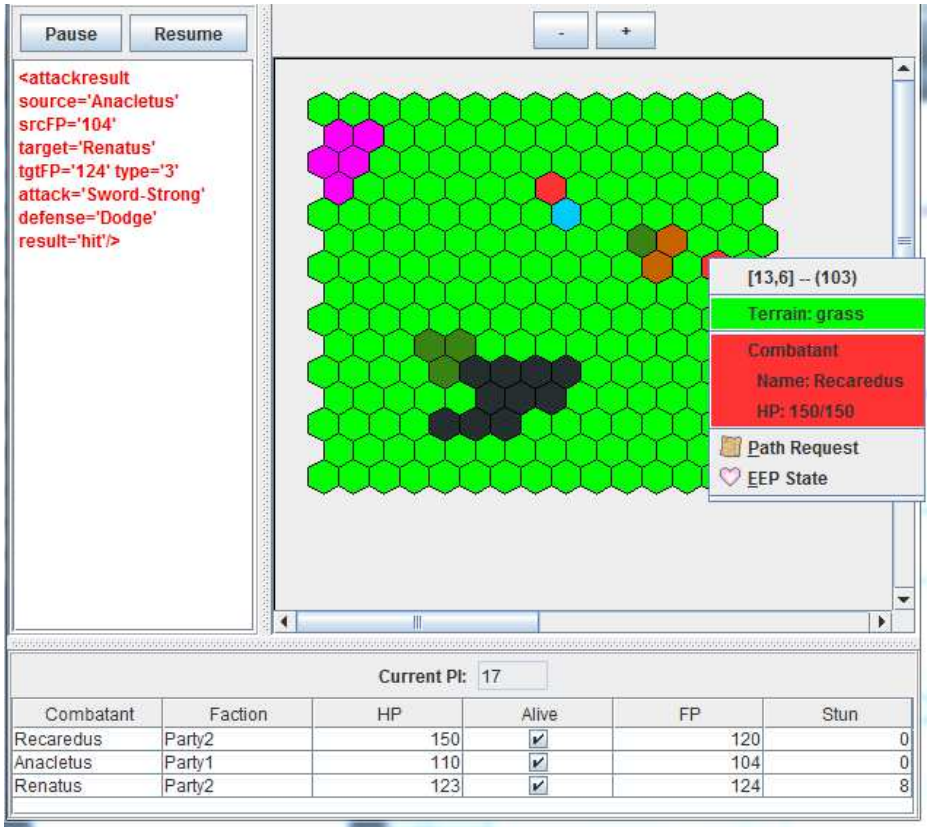


Figure 6.3: vBattle game state. At a specific instant, the game is described by the information of the battlefield, and all the states of the Combatants.

6.1.6 vBattle Implementation

The Core Rules of the vBattle are complex enough to represent the issues arising in a typical commercial video game. The implementation of the vBattle is guided by the final research purpose. The architecture of the vBattle game engine is based on components, so it spares the performance in order to improve the modularity.

The vBattle Framework must meet different requirements to implement the Core Rules.

Requirements from Core Rules

The requirements related to the game mechanics are:

1. *Different Tournament Configurations*: A Tournament refers here to a combination of different battlefield maps, goals and scenario set ups.

2. *Faction and Combatant Controllers*: It includes different combatants teamed up in factions. Both Combatants and Factions are managed by different controllers, and equipped by different action sets.

3. *Discrete Event Simulator*: The core of the game dynamics is based on the Event Sequence and the declaration-execution of actions. All the actions in the game are first declared, then prepared during a period of time for their, and finally executed.

4. *Different Action Rules*: The execution of the actions results in the amount of energy they consume and in the effects they produce on the environment (Combatants and scenario). These rules must be modular, thus they can be easily changed without producing variations in the main flow of the engine.

5. *Battlefield Objects*: The scenario is described in a 2D hexagonal grid, on each cell of the grid there can be different objects that may have visual occlusion properties, movement restrictions, or any other characteristic that could be relevant for the goals of the scenario.

Non-Functional Requirements

Aside from the functional requirements that are produced by the different elements that compose the Core Rules of the vBattle, there are also other restrictions that should be included, these are:

1. *Scalability*: The requirements in computation resources should deal with the performance issues that arise when the complexity of the environment increases (adding more combatants or factions).

2. *Monitoring Capabilities*: The simulations must be logged and visualized, so they can be analyzed for research purposes.

3. *Modular Controllers*: The vBattle framework is intended to be used as workbench for computation intelligence techniques further from the scope of this thesis. Thus, the purpose of this framework is also to provide an environment to test different algorithms applied to video game AI controllers, from basic capabilities, like navigation to complex interaction like group / coalition formation or coordinated planning.

Framework Analysis

Considering these requirements, we created the following analytic design. The vBattle Framework has been divided into different, independent components. Each of

these components not only must deal with certain requirements, but also with the other components in order to build a complete framework that fulfills all of the requirements.

The main features of the vBattle Framework are:

1. *A distributed system.* The communication is based on a message-oriented middleware, that connects all the components.

2. *Modular controllers for the behaviors.* The two levels of reasoning are managed by different components that can be attached, improved, or modified if necessary.

3. *Detached user interface.* It will have different interfaces to show the simulations and the results. These interfaces can be detached completely and leave the engine running without the burden of the graphical interface, skipping at the same time the refresh latency problem that could slow down the simulations.

4. *Configurable components.* The battles, maps and combatants are configurable through plain text files. This feature enables the configuration and modification of the simulations.

5. *Component Roles.* Each of the components that builds the framework has its own role, keeping the coupling of the components very low.

The general architecture created for the vBattle Framework can be described by the use of four different types of components (see Figure 6.4): (1) core system components, that enclose the scenario management, the game engine and the tournament creation features; (2) AI control components, that deal with the factions and combatants' behaviors; (3) component configuration descriptions, that set up the components to certain simulation profiles (tournament rules, scenario description, combatants profiles, ...); and (4) the user's interfaces, that show to the user the scenario information for analysis and visualization purposes.

Framework Design

The vBattle Framework is organized as a distributed architecture with different components that exchange information. The communication among the components is based on a *Message-oriented Middleware* that dispatches messages according to the diffusion and response needs of the components. The vBattle Core Rules components are connected to this middleware. These main components (see Figure 6.5) are:

1. **Tournament Manager (vBTM)**: This is the starting component that builds the different scenarios according to the specific goals, factions and maps involved. It notifies the Scenario Manager and the Combat Engine which map and combatants are involved in the current encounter (called "*battle*"). This component is in charge of the schedule and evaluation of a series of simulations. It runs from the first simulation

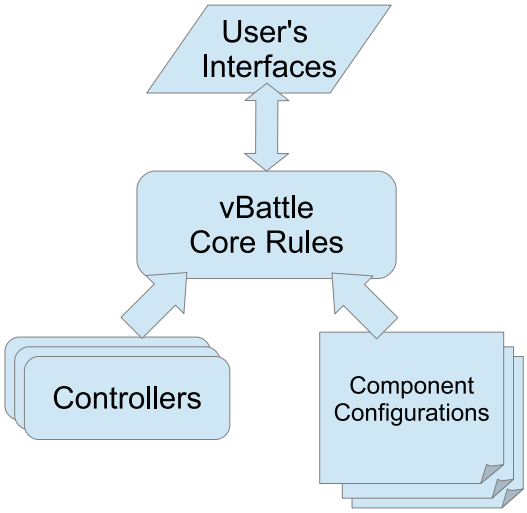


Figure 6.4: vBattle Framework Components: Analysis View. The Components of the framework are decomposed in the Core Rule components, the User's Interfaces, the Faction and Combatant Controller and the Component Configuration Descriptions

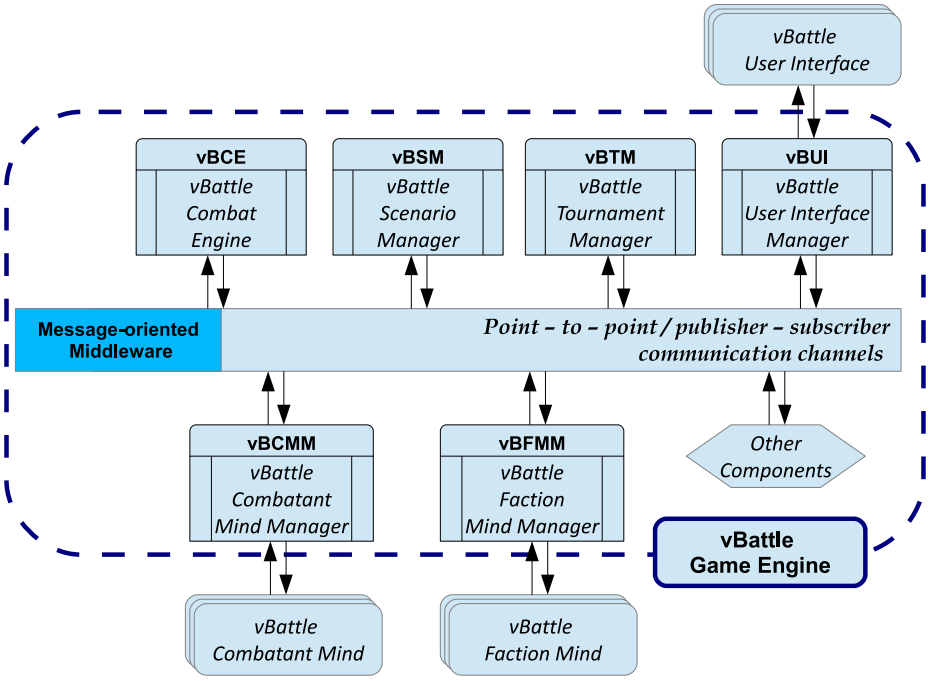


Figure 6.5: vBattle Framework Components: Design View

to the end of the last, it also tells all the other components when the current simulation begins and ends.

2. **Combat Engine (vBCE)**: The application of the Core Rules is mostly implemented by this component. It handles the Discrete Event Simulation by requesting the dif-

ferent components any information required to carry out the simulation. The vBCE keeps the information about the Combatants and Factions and evaluates the combat conclusion criteria. That information allows it to notify the vBTM at the end of the simulation. The vBCE also announced the result of the battle.

3. **Scenario Manager (vBSM)**: This component handles the battlefield. Here is where the movement rules and restrictions are applied. The vBSM also keeps the information about the tiles on the map, the conditions and characteristics of the key zones and the special information related to the objects in the scenario. Moreover, the Line-of-Sight and the Field-of-View of the Combatants are attached to the Scenario representation.

4. **Faction Mind Manager (vBFMM)**: It assigns the different Faction Leader controllers. The vBFMM also manages the composition of the Factions and the Combatants that are in each of them. In addition to that, the vBFMM handles the orders issued to the Combatants by their FL and applying the rules of command and propagation of these orders. The *Faction Minds* controlled by the vBFMM are asynchronous. They can issue orders when they need to, but the application of these orders is made synchronously in the *Combatant Minds* as part of their decision-making process.

5. **Combatant Mind Manager (vBCMM)**: Every Combatant from each Faction has its own controller. The *Combatant Mind* controllers establish the behavior and the decision-making process of the Combatant that they control. All of these controllers are handled by the vBCMM. This component is the interface between the Combatant Minds and the rest of the components.The vBCMM asks for the actions to be done by each *Combatant Mind* when the vBCE requests it. It also provides the information about the battlefield, keeping the information about the scenario (their Field-of-View and the events related to it, ...) available to each Combatant.

6. **User Interface (vBUI)**: The representation and visualization of the simulations made on the vBCE can be logged and displayed in different components.There are different kinds of *User Interfaces* that receive the similation information and display it, from 3D graphical environments to text-based or even databases. The vBUI manages the information provided to the different user interfaces and handles the requests that these component produce, while it asks for more detailed information about some of the elements of the game.

7. *Other Components*: The vBattle Framework is open to include more components for further improvements and requirements.

All of these components are communicated through a *Message-oriented Middleware* that is handled by the *Message Broker*. The architecture is distributed in such way that each of the components can be executed in different nodes of computation.

The information is shared using a set of formatted messages that are created by the different components. There are four types of messages that can be sent and received:

1. *Ask*: they are point-to-point bidirectional exchange messages of the asking-answering type. These messages are used when the actions need to be verified by both components, so they allow the presence of the necessary negotiation mechanism.

> The vBCE *asks* to the vBCMM for the next action to be done for a certain Combatant controlled by a particular CombatantMind. ■

2. *Commit*: They are point-to-point unidirectional messages that expect neither a proper answer nor a confirmation from the message target. They are used to inform to some of the components about the changes on the state of the message source component.

> The vBCE informs (*Commit*) to the vBTM about the finalization of a current battle after the successful closing of all of the controllers involved in that current battle. ■

3. *Request*: They are one-to-many bidirectional messages, used for the initialization of different components at the same time. When a component asks for the readiness of another set of components, it must waits until the last confirmation is received, in order to validate the request or not.

> The vBTM *requests* the vBCMM and vBFMM to inform it about the availability of the different controllers (for Combatants and Faction) involved in the next battle before to start the configuration cycle for this combat. ■

4. *Advertise*: This is a broadcast message used to distribute the progression of the simulation among all (or a set) of the components. The component sending this type of message do not expect any response, like in the case of the "commit" messages, but they describe the changes in the state of the environment.

> The vBCE *advertises* to the all of the components whether a certain attack launched by Combatant was successful or not. ■

Thanks to these messages, the vBattle Framework composes all the information exchanged among the different components during the execution of the different scenarios, battles and tournaments. This message exchange schema has proven to be very effective for the distribution of heterogeneous environments. Furthermore, it leaves the framework open for the inclusion of additional components, programmed in different platforms.

6.2 Integration Proof-of-Concept

Although each of the components of the AGCBAR Architecture has been independently validated (Chapter 4 evaluates the emotional model and Chapter 5 compares the learning strategies), the objective of the architecture itself can only be achieved when all these components work together in a close-to-real complex scenario.

The goal of this final chapter is to demonstrate, in an integrated system, the implementation of the whole components of the architecture. The main objective is to analyze the controller–development effort needed to implement a set of controllers, while showing the benefits of the integrated architecture.

The implementation of the AGCBAR Architecture including the vBattle should:

1. Exploit the emotional features provided by the Emotional Engine.
2. Use off-line learned control strategies for the Combatants.
3. Coordinate all the components of the architecture on top of a game engine with a realistic level of detail.

6.2.1 vBattle Scenario Implementation

For the proof-of-concept proposed in this thesis, we include a subset of the features and rules of the vBattle in order to present the results of the analysis of the scenario in a clear, well-organized manner.

One AI Layer Subset

The decision-making architecture of the vBattle is divided in two layers: a first one for the high-level planning represented by the *Faction Minds* and a second layer of actions oriented to directly control the Combatants' behavior on the battlefield.

For our configuration of the scenarios in this experimentation, we decided to include only the low-level actions only. The reason behind this is that we want to study the construction of the controllers of the Combatants and the adaptation of these controllers to the *Mood Dynamics* concern of the AGCBAR Architecture.

Thus, *we analyze the* ***Combatant Minds*** *and the resources needed to build and adapt them using the AGCBAR Architecture.* As result, the actions performed in the environment are only decided by the Combatants. They will try then to achieve the objectives

of the scenario without the supervision of the Faction Leader (which would be controlled by a Faction Mind).

Contrast Static Combatant Mind

We provided a *baseline contrast controller* implementation. This implementation is used as the initial description of the development of the different static controllers across the integration of the architecture. For this controller, we construct three different Behavior Trees, one for each Combatant action type (i.e. level of decision): Combat, Defense and Movement. These Behavior Trees, see Figure 6.6, represent the strategies needed by the Combatants, so they can provide the corresponding action at the different instants of the simulation. These strategies are:

- *The Movement Strategy* is designed to advance toward the enemies, looking for the nearest enemy.
- *The Combat Strategy* is divided in three branches: When the Combatant is far from his current target (nearest enemy) he runs, when he is exhausted (with few remaining FPs) he rests. In any other case, he will attack an enemy within range, choosing randomly the type of attack among the different available ones he has.
- *The Defense Strategy* is used when the Combatant managed by the contrast controller is attacked. In this case, he will choose the type of defense to be used randomly, except when he is exhausted, in which case he will always make a "Soft Block" (because this is the defense that consumes less FPs).

Scenario Goals Implemented

From the set of possible scenarios that vBattle can present, we chose the most general objective in order to leave the Combatants complete freedom of operation. Here, we picked "Kill'em all" scenarios. The goal of these scenarios is trying to kill all the enemies on the battlefield. This objective is easy to identify and to implement by the Combatants because it may not need of too much coordination, and it also allows the Combatants to choose their own strategy and behavior.

Scenario and Battlefield Configuration

For the experiment, we choose to create a simple scenario (see Figure 6.7) with the following elements on the battlefield:

- One deploy zone on the left-hand side of the battlefield and another one on the right-hand side.

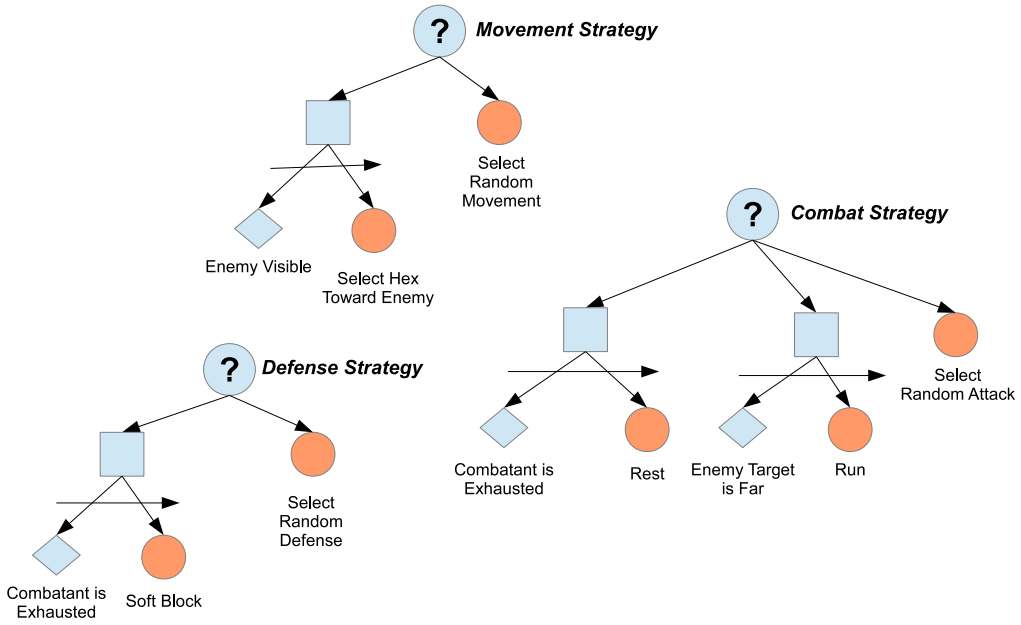


Figure 6.6: vBattle Proof-of-Concept: Contrast Static Controllers. A Combatant decision-making process is divided in three levels: Movement, Combat and Defense.

- A flee zone on the upper side of the battlefield.
- A sparse distribution of fixed objects (trees and rocks).
- A uniform terrain type (no swamp or rough terrains).

For the factions configuration, we set the same Combatants for the two teams (the same combatant profiles). The *Party1* (blue team) begins on the left-hand side and the *Party2* (red team) initially spawns its combatants on the right-hand side of the battlefield.

6.2.2 AGCBAR Architecture Integration Issues

According to the AGCBAR Architecture definition, we must implement a set of elements in order to satisfy the integration of the model. As we described previously, both concerns of the architecture are going to be implemented in this experiment. Therefore, we must describe all the elements of the AGCBAR Architecture. In this section, we specify the components of the architecture that we applied in the vBattle.

Architectural Dictionaries for vBattle Integration

Taking the elements described in the structural view of the architecture in Section 3.2, the first elements to be defined are the Architectural Dictionaries. See the Appendix D.1 for their complete description.

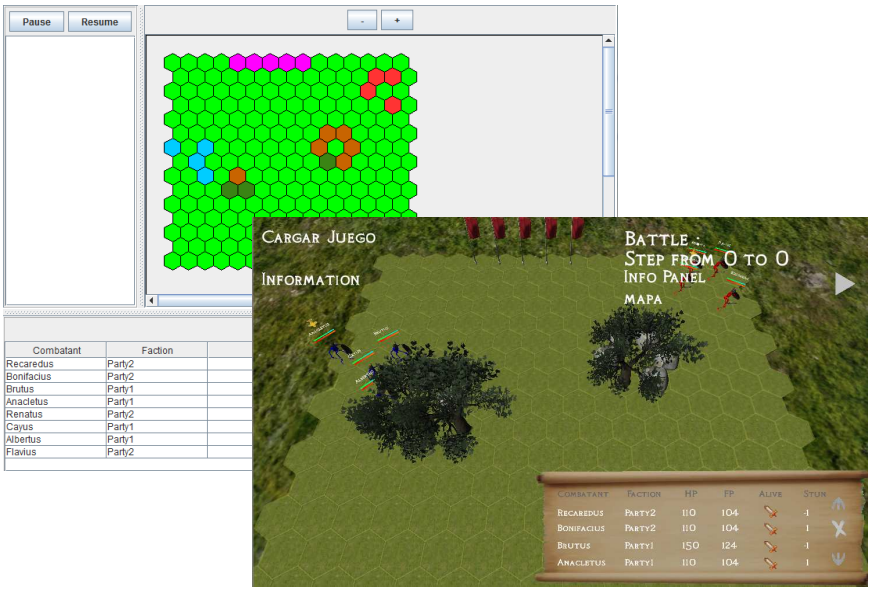


Figure 6.7: vBattle Proof-of-Concept: Scenario Setup. It shows the initial distribution of the factions on the battlefield.

In this experiment, we will identify five different event classes that are relevant for the *Mood Dynamics* concern. These classes are related to the actions of the Combatants and to their possible results. Thus, the `Move` event class identifies the actual change of position of a Combatant that could be relevant for the sense of faction-support and that it, consequentially, could influence the mood of certain characters. Also, the `Dead` or `Fled` event classes are very important for the possible prompted emotions of the Combatants (worrying the dead/fled Combatant's allies and inspiring his foes).

Also related with the *Mood Dynamics* concern, we will describe here four different moods for the characters of this scenario. They represent the four Combatants' goals described in Section 6.1.4. The `Normal` mood coincides with the *Obey Orders* of the Combatant, the `Angry` mood describes the Combatant's tendency to *Pursuit the Enemies*, the `Afraid` mood describes the Combatant's goal of *Stand and Fight* and the `Broken` mood identifies the *Flee and Survive* objective.

The *Environment* and the *Actions Dictionaries* are used to provide the game state information and the set of actions for both, the static and the automatically created controllers.

The *Goals Dictionary*, used in the *Automatic Controller Creation* concern, describes four goals for the learning process: Preserving Energy, Maximizing the Enemy's Damage, Minimizing the one's Own Damage and Running Away. These conceptual goals will be aligned with the four moods of the Moods Dictionary.

Engines for vBattle Integration

After presenting the dictionary description, we will define the Engines. In our case, the vBattle is the Game Engine. It will provide the emotional events, E_i, described in the Events Dictionary. It will also identify the current state of the environment, $\vec{S_i}$, so that the Learning Engine, and lately the Current Strategy, can select the next action, A_i, to be executed. The Emotional Engine is implemented by the *EEP Model* and the Learning Engine by the *WEREWoLF-DE* (implementation of the WoLF learners with DE evolutionary controllers).

Interfaces for vBattle Integration

The implementation of the Engines carries along the implementation of the interfaces. Thus, the *EEP Model*, implements the *Event Interface* (named Event Builder in the EEP Model architecture) and the *Mood Interface* (Mood Tagger in the EEP Model).

In the case of the Event Builder, we have established a relationship between the Event Classes from the Events Dictionary and the Extended Event Classes used in the EEP Model obtained for the analysis of the events prompted in the game engine. The Event Builder analyzes the events with a specific rate (each 10 time steps of the game). In this analysis, we can obtain the consequences of the attacks included in the `Wound` events prompted after the `Attack` events. We also analyze the proximity of allies and enemies in order to produce consequences related with the `Move` events. And finally, the `Dead` and `Fled` events are considered specially important to prompt the appropriate consequences. The Actions, Consequences and Objects dictionaries used in the EEP Model are described in Appendix D.2.

The WEREWoLF-DE implements the *Actions, Environment States and Goals Interfaces*. The *Action Interface* transforms the actions described in the Combatant Profile into the abstract Actions specified in the Actions Dictionary (for instance, it associates the specific "Fist" attack to the `Soft Attack` described in the dictionary). The *Environment States Interface* adapts the vector of the environment state parameters to a discrete representation of these parameters in order for it to adapt the environment state to the selected Learning Engine. Thus, for instance, when we create the strategies for the Combat Action level of decision of the Combatants, we use the following environment state parameters:

- Combatant Current HPs (*cHP*),
- Combatant Current FPs (*cFP*),
- Number of allies within 5 hex radius from the Combatant (*AlliesIn5*),
- Number of allies within 5 hex radius from the Combatant (*EnemiesIn5*),
- Distance To Nearest Enemy (*DistanceToEnemies*),

- Distance To Nearest Flee Zone (*DistanceToFlee*),
- Last Damage Rank (*DamageRank*).

Finally, the *Goals Interface* establishes the correspondence between a Goal described in the Goals Dictionary and the reward and fitness functions provided to the Learning Engine for the optimization process.

Processes for vBattle Integration

When the processes of the AGCBAR Architecture are implemented. The *Goal Selection* is done through an iterative process where we can select each of the Goals and evaluate the environment, so we can build up a different strategy for each goal. The *Strategy Selection* is made accordingly to the mapping between strategy and mood. As we created one strategy for each Goal, we associated each of these goals to a Mood, so we can select the associated strategy for each mood.

6.2.3 Evaluation Criteria Overview

There are several aspects to be considered as a measure for comparing the results derived from the AGCBAR Architecture. Some comparative results has been presented in the previous chapters where, as an isolated component, we have (1) tested the performance of the learning methods against the hand-coded alternatives, and (2) evaluated the behavior derived from the modeling of different mood states and emotional responses.

The final twofold aspects to demonstrate are:

1. The integration of the separate components: Verified as a complete system in execution.
2. The benefits in the game production by the reduction of programming costs: Evaluated by the software engineering metrics of the hand-coded alternatives and the time and resources spent by the assisted mechanisms that are part of the AGCBAR Architecture.

Mood Dynamics & Number of Controllers

The inclusion of the *Mood Dynamics* concern in the features provided in a real video game scenario implies the development of several extra controllers. This is the price to pay if we choose to show more realistic behaviors. Figure 6.8 presents, in a schematic representation, the increase in the number of controllers when *Mood Dynamics* are considered, but also depicts the relative probability of reusing part of the code among these different controllers.

In terms of code development, character controllers may share part of the implementation. It is not easy to just measure the percentage of reusability of the code.

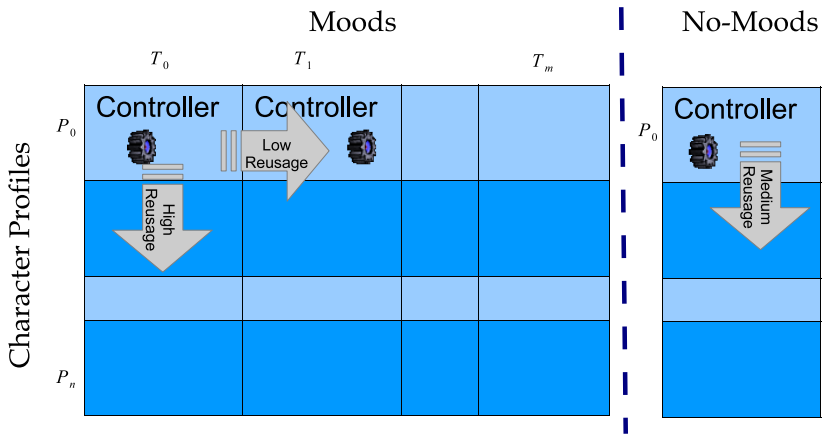


Figure 6.8: *Mood Dynamics* & Number of controllers ratio. When the *Mood Dynamics* concern is included the number of controllers is proportional to the number of moods that we describe for each character profile. The reusage of the code is significant when sharing implementation among profiles.

Nevertheless, when including extra controllers for the different moods of a given character, the controllers among the different moods have a moderate reusability, because the Moods are related to the Goals (as they are define in the AGCBAR Architecture), while controllers for a particular mood are likely to be quite similar among different characters (at least for some moods, for example, "Broken/Scared" moods).

6.2.4 Development Costs Evaluation

In order to evaluate the implementation effort of this experiment in the vBattle, we analyzed the development cost of a game controller for a Combatant. This controller (*CombatantMind*), as we said previously, has to provide the actions to the Game Engine in the three different strategy levels:

1. *Movement*: when the vBCE asks for the next movement position for a given Combatant. This Combatant has to evaluate his neighborhood in the given scenario and decide which is his best movement according to his actual goal.

2. *Combat*: as we describe in the vBattle design, the Combatants decide their next action during the Discrete Event Sequence (DES), so they overlap their actions along the time, breaking the turn-based sequence. Each time that a Combatant finishes one action, the vBCE requests from him the next action to be performed, scheduling this action in the event sequence. To be able to decide which action to perform next, the Combatant has to analyze the environment state, $\vec{S_i}$, and send his action,A_i to the vBCE.

3. *Defense*: every time that a Combatant is the target of an offensive action from an enemy Combatant, he must decide which defense to apply against this attack.

In addition to this basic implementation, we included the desired feature of the *Mood Dynamics*. In order to do so, we implemented a mechanism to create emotions. As we said in Chapter 4, the creation of feasible emotional models is complex, so we adopted the EEP Model (any other emotional model that provides a coherent reaction to the events could be used as well). Therefore, we only had to provide the strategies for each of the Moods described in the Moods Dictionary.In the previous section, we described four different Moods (`NORMAL`, `ANGRY`, `AFRAID` and `BROKEN`). We also had to create four strategies with three levels of decision. Thus, in this case, we have to implemented 12 different strategies per character profile.

To analyze the code needed for the integration of the different Strategies with the Moods we chose two baseline implementations: static controllers programmed with the Behavior Trees, and learned strategies extracted from the WEREWoLF-DE engine. In each of the cases, there is a portion of the code, that we called *Structural Code*, which is necessary for each implementation regardless of the number of different strategies we want to create. Over this structural code we built the *Specific Code* which represents the strategy in itself. These pieces of code are the decision process (in the case of the static controllers) and the goal description and action/environment codification (in the case of the reinforcement learning controllers).

The *Specific Code* for the static controllers has been made by altering the baseline implementation and by changing the strategies corresponding to the specific Goal represented by each Mood. The Movement strategy is adapted according to the Moods as follows:

- `Angry`: the same strategy because the Combatant wants to reach his nearest enemy.
- `Afraid`: if the Combatant has more than two allies near his current position, he moves toward his nearest enemy. If not the Combatant stands in his current position.
- `Broken`: the Combatant moves toward the nearest Flee Zone on the battlefield.

Depending on the Combatant's Moods, we modify his Combat Strategy as follows:

- `Angry`: the Combatant makes the same actions as in the default (`Normal`) strategy, but always taking the decision of launching the "Hard Attack", so he try to inflict as much damage as he can.
- `Afraid`: in this mood the Combatant changes his strategy a little, resting more often (increasing the threshold that represents the exhaustion). His attack selection also changes to a strategy that has more probability of launching a "Soft Attack" or "Quick Attack" than any other attack options.
- `Broken`: the desire for surviving forces the Combatant to choosing the "Run" action in any case.

Finally, in order to adapt the Defense Strategy to the Moods we altered the default strategy:

- `Angry`: the Defense is chosen between "Soft-Block" or "No-Defense" because the Combatant does not want to waste time parrying the incoming attacks.

- `Afraid`: the Combatant always decides (and as long as he is not exhausted) to use a "Full Block" in order to maximize his probabilities of avoiding damage.

- `Broken`: in this case we apply the same strategy as in the `Angry` mood, but for the opposite reason, the Combatant wants to run away.

The automatically created controllers are built with a *Specific Code* that describes the goals in terms of a reward function and a fitness function. The fitness function is always the same for all the Mood. It is calculated as the average final reward obtained by the Combatant in each episode. The reward function assigned to each Goal is constructed for representing the objectives described in Appendix D.1. For instance, in the reward function used to build the Combat Strategy that represents the Maximize Enemy Damage goal, we included the damage caused to the enemy with the last attack, the energy consumed, ...

Development Cost Metric

In order for us to quantify the amount of work required for the development of these two alternatives in our framework (where we need different strategies for the four different Moods specified in the Architectural Dictionaries), we will use the *Development Effort* specified by the COCOMO-II Methodology [9][6].

Also, for us to be able to provide the estimation of the effort and to extrapolate the numbers for a large of set of potential controllers, we have recorded the lines of code of the present vBattle implementation (Table 6.8).

In Table 6.9, we present the effort required to developing the controllers, measured in Man-Month. We calculated its cost taking into account the fact that the Behavior Trees can be easily reused, so we estimated the development cost by applying different re-usability factors (from 0% to 75%). In Figure 6.9, we present the evolution of the development cost as we increase the number of controllers.

[6]COCOMO stands for *Constructive Cost Model*, it is an estimation model that measures the cost of software development. COCOMO was first published by Boehm in 1981, and later updated in 1995. COCOMO takes the number of lines of code as input, as well as other software development characteristics, to estimate both the required Man-Month and Development Time. COCOMO provides three different levels for the model: Basic, Intermediate and Detailed. Basic COCOMO does not consider many project attributes (cost drivers) that are incorporated in the other two levels.

Package	**LoC**	
vBTM	688	
vBCE	1460	
vBSM	289	vBattle **Game Engine. Core Components**
vBFMM	503	
vBCMM	1311	
vBUI	1729	
Utils	2649	
BT (Structural)	656	
RL (Structural)	1565	vBattle **Combatant Minds**
BT (Average per Controller)	167	
RL (Average per Controller)	24	

Table 6.8: vBattle Proof-of-Concept: Lines of Code of the different vBattle packages. The table also includes the *Structural Code* and the average *Specific Code* per Controller needed to build the Combatant Minds for the Behavior Trees (BT) and the WEREWoLF-DE (RL) alternatives.

	Behavior Tree 0%	**Behavior Tree** 75%	**Automatic Controller Creation**
Structural Code	1,54	1,54	3,84
3 Controllers	2,80	1,85	4,02
12 Controllers	6,71	2,79	4,59
24 Controllers	12,11	4,09	5,34
50 Controllers	24,17	6,94	6,98
100 Controllers	48,14	12,57	10,19
200 Controllers	97,70	24,17	16,76

Table 6.9: vBattle Proof-of-Concept: COCOMO Development Effort metric to develop different number of controllers. The Behavior Trees can be easily reused,

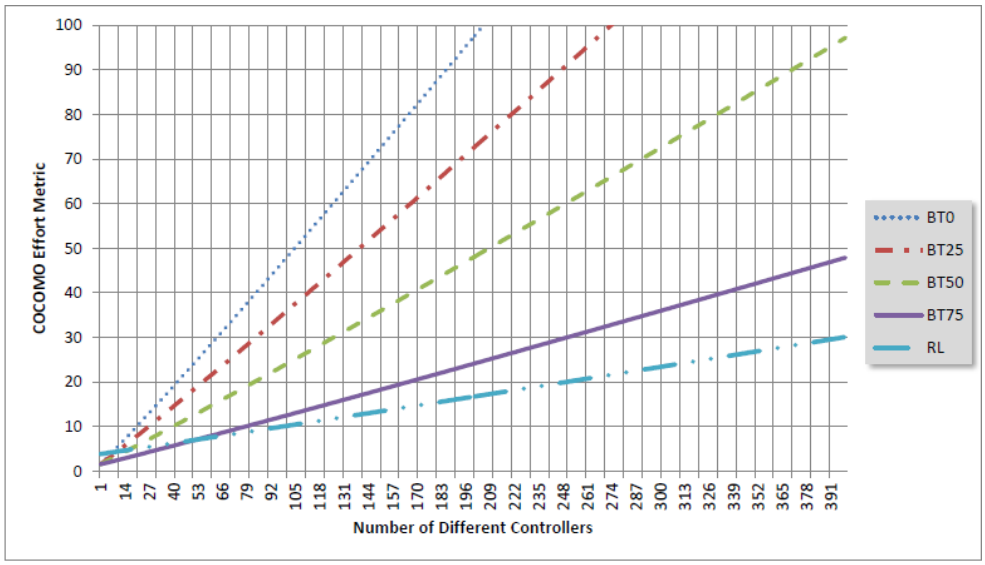


Figure 6.9: vBattle Proof-of-Concept: COCOMO Development Effort Metric (in Man-Month). The Behavior Trees are represented taking a code re-usability of 0% (BT0), 25% (BT25), 50% (BT50) and 75% (BT75) compared with the Reinforcement Learning (RL) alternative

6.3 Discussion

The present chapter illustrates the implementation of the AGCBAR Architecture in a video game engine. We created the implementation of the video game vBattle [25] that meets two objectives: 1) it is complex enough to represent the standard products of this software industry: it has all the components of a video game, it is open and modular to extend and to modify it; and, 2) it applies both of the AGCBAR Architecture concerns: the *Mood Dynamics* and the *Automatic Controller Creation*, with all the elements needed to implement these concerns.

Thereafter, we will evaluate the different behaviors, models and strategies resultant from the application of the aspects present in the architecture. Hence, we analyzed the range of possible behaviors that can be obtained from the application of the AGCBAR Architecture.

The *Mood Dynamics* concern increases the believability, it provides the adequate behavior consequent with the environment events. The counterpart of this feature is the increase in the number of controllers, due to the need of providing a different behavior to each of the characters' moods.

On the other hand, the *Automatic Controller Creation* automatically produces different strategies within the complex environment. This module adapts the strategies through the exploration providing two interesting outcomes: 1) the different strategies, and 2) the game bug exploit discovery. The inclusion of this concern in the development of a video game contributes to decrease the controllers development cost. This is interesting

from a cost-efficient point of view because, due to the *Mood Dynamics* inclusion, we can face a significant increase in the number of controllers.

Moods and Emotions

One of the main objectives of this thesis is to provide richer and more believable responses from the characters in the video games. The Emotional Engine included in the AGCBAR Architecture contributes to this purpose by including the emotions, moods and affects within the game scene. Thanks to that, the characters' controllers are focused on their actions in the environment, according to their theoretical mood. This component of the architecture handles the emotional representation and the mood changing with a minimal programming overhead.

- **Pros:** the controllers only have to resolve the actions and the emotional engine and its configuration resolves the emotional interaction.
- **Cons:** the emotional engines probably need the specification of the relevant environmental events, this need is a key factor when we are to develop the mechanism that generates the emotional events. This problem can be solved by using models that can exploit the knowledge within the Conceptual Dictionaries (as described in Section 4.3.4), like taxonomies and inference.

Different Strategies

The presence of the Learning Engine enables the existence of mechanisms that produce different types of basic controllers that can be used in the game scenarios. These controllers can be selected according to different criteria: they can have different challenge rate, allowing them to use more or less learn during time, or they can have different strategy basis, giving different representations of the actions or state spaces so they can explore the environment differently.

In our case, we used the WEREWoLF algorithm with a population of 10 individuals evolved after 100 episodes. We observed that the different individuals of different generations prefer different actions in each state, such actions are selected according to the acquired fitness.

- **Pros:** we can have different controllers that can be used in the same game with enough variation for the same character to enrich his behavior and robust enough to work reasonably well.
- **Cons:** the Learning Engine should ensure the robustness of the strategies created, and of some of the controllers created, mainly throughout the early stages of the learning process. During these stages, they could be not too smart, and could perform some completely unreasonable actions. Hence, if we try to use this mechanism for the

creation of a broad spectrum of controllers, we must also have a mechanism that validates and/or constrains the actions, so they are not completely incoherent.

Moreover, if we analyze the number of different characters that are present in the last generation games, we can easily observe an increasing interest for the character controllers development support. For instance, Assassins Creed III™[7] has 10 different types of enemies, but, Elders' Scroll: Skyrim™[8] has more than 350. Other games, like Fable 3™[9] have over 100 different characters. The amount of characters (especially if we multiply the number of controllers adding the moods to the environment simulation) is increasing. Figure 6.9 presents a possible evolution in the development effort needed to achieving this number of controllers. Therefore, the *Automatic Controller Creation* is an interesting alternative to support the development of new video games with a large amount of controllers.

Bug Discovery

Also, we came upon the fact that the already created controllers exploit some of the weaknesses that the static controllers have. Additionally, we discovered certain problems with the targeting and movement systems of these contrast controllers. This finding enabled us to debug and polish the behavior trees so they have less bugs.

Thereby, the *Automatic Controller Creation* contributes to the development of a better system of new controllers speeding up the debugging process of the traditional ones. Also, these new controllers can also be used for testing scenarios that have a large variety of rules and features, since they explore the environment freely during their learning process.

[7] http://assassinscreed.ubi.com/ac3/en-GB/games/assassins-creed-3/index.aspx

[8] http://www.elderscrolls.com/skyrim

[9] http://lionhead.com/fable-3/

Chapter 7

Conclusion

> It's more fun to arrive a conclusion than to justify it.
>
> Malcom Forbes

In this thesis we introduced an architecture that addresses the issue of improving the development of next generation video games, where the behavior of the characters can be mood-driven and the number and diversity of the controllers can also be larger than in previous developments. In this last chapter, we will summarize and discuss the main contributions and the proposed future lines of work.

7.1 Contributions

The objectives presented in the Section 1.2 of this thesis are mainly focused on the application of software models to video game developments.

7.1.1 Contributions Overview

To this end, we adopted different fields of study in order to carry out our research, while trying to improve the players' experience in video games beyond the graphical elements:

1. We embraced the affective component of the characters as a potential major improvement, with the intention of adding a deeper sense of realism into many video game scenes. For that purpose, we drew upon work from the field of cognitive psychology so as to construct a computational model that could be applied to this particular software. The result of our research and the findings was the creation of the *EEP Model* (described in Chapter 4).

2. We found that the creation of controllers in many video games is very time-consuming, and error-prone, and the resulting controllers are not fully challenging for experienced player. Thus, we concluded that the static solutions for the characters' behaviors could be improved thanks to the use of different learning mechanisms that allows us to build them up. The automatic creation of contents, and in our case character' controllers, is a trending topic in the game industry. Thereby, we studied the soft computing techniques suitable for this environment. And finally, we developed the *WEREWoLF Algorithm* (detailed in Chapter 5).

3. The aforementioned models alone can be applied to some video game developments, but their application became much easier if they are handled as part of an architecture. This architecture defines the communication between the different models proposed in this thesis and the video game engines. Thus, we designed the *AGCBAR Architecture* as a global framework that supports our proposed solutions (presented in Chapter 3).

7.1.2 Contribution in the Emotional Modeling Context

Emotional modeling is a broad topic concerning different disciplines and potential applications. In this thesis we focused our attention on the exploitation of this modeling applied to video games. In order to achieve the purpose of this thesis, we have followed a series of preliminary steps:

1. Identification of the requirements needed for the emotional simulation in the video game context.

2. Analysis of the existing emotional models applied to virtual characters.

3. Study of the cognitive psychology theories of:

   (a) Ortony, et al. OCC model for structure analysis of the emotions.

   (b) McRae and Costa Big Five Factor for personality.

   (c) Mehrabian and Russel PAD Temperament model for representing moods and emotions.

This thesis contributes to the existing research on emotional modeling providing a new model, named *EEP*. This model shows the following characteristics:

- *EEP* defines a series of Mood Tags to label different Mood states.

- *EEP* is based on a mathematical formalism, named Mood Vector Space (MVS), an extension of a Hilbert Space with the corresponding operations that provide Emotion combinations and Mood Dynamics.

- *EEP* includes a structured analysis of the emotions produced by the events perceived by the characters.

- *EEP* can describe the personality profiles of the characters and translate them into the initial and central tendency of the characters' mood.

Henceforth, the *EEP Model* has been evaluated as an integrated component in a commercial video game engine. The evaluation has been carried out by testing the model under two different scenarios:

1. A combat scenario in which moods influence the behavior of the different characters. These behaviors are implemented as different controllers managing the characters when they are in a corresponding mood.

2. A storytelling scenario in which the plot evolves according to the mood transitions of a series of NPC characters populating the environment. The moods in this scenario change as a result of the actions of the player and the relationship among the characters.

Finally, we have conducted a comparative analysis, in terms of features and expressive capabilities between our *EEP Model* and the existing state of the art references, in particular the *EMA* and the *ALMA* models.

7.1.3 Contribution in the Learning Controller Context

The design and coding of character controllers is a time-consuming and error-prone activity, to which the game developers should devote a significant effort across the game design and implementation. The mechanisms to produce reliable character control codes, in an automatic way, are of great interest for the industry.

This thesis has proposed a new strategy to craft character controllers for combat scenarios by means of the following outline:

1. A preliminary definition of a set of reference controllers. These controllers can be either hand-coded versions or the result of a previous automatic character controller creation process.

2. An iterative training procedure for the automatic creation of new controllers through the competition against a set of reference controllers.

3. The extraction of the learned strategy as a new control package.

The core element in this training procedure involves a learning algorithm for character controllers. This thesis proposes an innovative algorithm, named *WEREWoLF*, based on the hybridization of two different computational intelligence techniques:

1. Evolutionary Computation. Two different evolutionary techniques have been tested, the Estimation of Distribution Algorithms (EDA) and the Differential Evolution (DE).

2. Reinforcement Learning. Two different temporal-difference alternatives has been considered, the SARSA and the WoLF algorithms.

An exhaustive experimental testing has been conducted, comparing the performance of the new proposed learning strategy (*WEREWoLF*) against the baseline reinforcement learning alternatives. The results obtained clearly show that the *WEREWoLF* configuration using a Differential Evolution strategy together with the WoLF learning algorithm, outperforms the rest of the configurations as well as the baseline algorithms. The experimental testing was supported by a statistical analysis, which allows us to assert the significance of each result with a robust confidence level.

7.1.4 Contribution in the Game Development Architectures Context

In order to integrate the two aforementioned contributions into a unified framework, this thesis has proposed a new component architecture that covers two concerns:

1. *Mood Dynamics*, which encloses the affective simulation component desired for the video game characters.

2. *Automatic Controller Creation*, that takes into account the creation process of the different strategies for the characters in the video game environments.

The new component architecture presented in this thesis has been named the *AGCBAR Architecture*, which stands for Automatic Game Controller Behaviors with Affect Responses. The different configuration of the components of the *AGCBAR Architecture* operates in two application phases:

1. Off-Line Strategy Generation. This phase supports the design and development stages of the video game production. It also addresses the *Automatic Controller Creation* concern.

2. Run-Time Action and Strategy Selection. This phase orchestrates the *Mood Dynamics* and the embodiment of the different automatically created controllers.

Finally, based on the *AGCBAR Architecture*, this thesis provides a complete environment for the research of many other aspects of the video game computational intelligence. The architecture allows for substituting any of its components in order to evaluate different techniques and models.

7.1.5 Other Contributions

During our experimentation, we stumbled upon some additional results that appeared throughout the development of the different models. They are not among primary objectives of our research but we believe that they are worth to be mentioned.

vBattle Framework

When we planned the experimentation of the whole architecture, we found major limitations when we tried to use some commercial video game frameworks. Usually, these were closed, not fully modular or, in the best cases, badly accessible. They lacked, in many cases, of a proper programming interface, which made the creation of the complex structures, algorithms and tests that we were going to need, impossible for us. Also, the experimentation of the complete architecture required an interface, adequate for all levels of the game engine. Hence, we created our own game engine. For that, we used the same techniques applied to the video game design, creating rules, components and modules to support the *AGCBAR Architecture* concerns. The vBattle framework is the result of this development. As will be said later in this Chapter, is already being extended to support for further experimentation. Nevertheless in the present state described here, it provides the necessary infrastructure for study of the models that we developed during the thesis.

Emotional Storytelling

The research carried out with the *EEP Model* shows that the initial application of reactive responses to the emotional events driven by the mood state can be used in some other ways. For example, the interactive storytelling is an interesting field of application of affective computing. Furthermore, the inclusion of emotional models consistent along the evolution of a story line could be applied for the creation of more coherent, dynamic and appealing plots. The first results shown in the experimentation of the *EEP Model* open new possibilities for further researches.

Learning Designer Role

After the development of the *WEREWoLF* algorithm experiments, we discussed the new role that is necessary for the correct application of the learning models in video games. Particularly, in the level design field, and aside from the core AI development, it seems that a Learning Designer could be needed. This role will need of specific knowledge to be able to carry out the correct layout of the reinforcement learning parameters. The state space representation and the reward function design are critical for the success of these models in a real development environment. This role requires the understanding of the rules of the game as well as of the machine learning techniques, so this special level designer can figure out the correct way to represent and model the world, and through the use of these techniques, to produce the correct controllers efficiently.

7.2 Future Work

This thesis leaves many lines of work open. We will define here a blueprint for future lines of investigation and development that we envision as upgrading of the work

presented here.

The *AGCBAR Architecture* presented in Chapter 3 bears the following open issues:

- The inclusion of the Learning Engine in the Run-Time phase is of particular interest because it can adapt the characters' behaviors during the game play. This feature must be supported by fast learning algorithms that can extract new strategies in real-time.
- The development of support tools that can help the design and implementation of video games that use the AGCBAR Architecture.
- The research of more complex, challenging scenarios, that will improve the generalization of the implementation of this architecture, and therefore, the development of different video games.

As we said previously, the introduction of the emotional models is a target for many video game developments, these models could increase the realism of the environments and behaviors, producing more engaging products. Our *EEP Model* has shown interesting results but it could still be developed in many aspects.

- The Taxonomy representation of the Conceptual Dictionaries could produce more flexible and structured representations of the character profiles. It could also increase the applicability of this model.
- The inclusion of the emotional models, as a storytelling guidance, could create new mechanisms to help designing the plots and the story lines of the quests and worlds of future video games.

The work made in the *WEREWoLF* could be refined and augmented in many ways,

- It could include some other core reinforcement learning algorithms, like Function Approximation, as basis for the hybrid combination.
- There are different mechanisms that could create a new hybridization model of the learning individuals.
- Preference learning could improve the learning process, providing an additional mechanism to tackle the complexity of the environment.

Finally, the vBattle framework still has a lot of work ahead. The research possibilities of this platform could be used in many branches of the AI in games, from controller creation to coordination scheduling. This kind of development could make the study of different techniques easier for the academic community. Furthermore, this framework is intended to be used in different teaching contexts, where the variety of techniques from the field of computational intelligence applied to video games can be tested and shown in a more appealing way.

7.3 Key Publications Produced during this Thesis

In this final summary, we would like to mention the most relevant publications produced throughout the development of this thesis.

- Peña L., Peña J.M., Ossowski S., Lucas S.M. *Learning and Evolving Combat Game Controllers*. IEEE Conf. on Computational Intelligence and Games. Granada 2012. [67]
- Peña L., Peña J.M., Ossowski S. *Representing Emotion and Mood States for Virtual Agents*. Multi-Agent System Technologies Conference. MATES 2011. Berlin 2011. [68]
- Peña L., Peña J.M., Ossowski S., Sanchez, J.A. *EEP – A Lightweight Emotional Model: Application to RPG Video Game Vharacters*. EEE Conf. on Computational Intelligence and Games. Seoul 2011. [65]
- Peña L., Peña J.M., Ossowski S., Herrero, P. *WereWoLF: Evolving Q-Learning for Stochastic Games*. World Automation Congress. Kobe 2010. [66]
- Peña L., Ossowski S., Peña J.M. *vBattle: A New Framework to Simulate Medium-Scale Battles in Individual-Per-Individual Basis*. IEEE Symp. on Computational Intelligence and Games. Milan 2009. [62]
- Peña L., LaTorre A., Peña J.M., Ossowski S. *Tentative Exploration on Reinforcement Learning Algorithms for Stochastic Rewards*. Hybrid Artificial Intelligent Systems Conference. HAIS 2009. Salamanca 2009. [63]

Appendix A

Terminology and Acronyms

A.1 Glossary

AAA class games Blockbusters video games with a big superproduction budget and high quality technical and detail aspects. 2, 3, 183

Architectural Dictionaries The dictionaries that describe the elements to use in the components of the AGCBAR Architecture. 53, 82, 126, 183

Battlefield General representation of the physical terrain where the battle runs. In the vBattle framework, it is decomposed in hexagonal cells. 145, 183, 205

Base Movement Rate Default time consumed by a Combatant when moving, in the vBattle framework, from one cell to another adjacent one. 145, 147, 151, 183

CD Conceptual Dictionaries. 88, 124, 183

Cell Hexagonal portion of terrain. In the vBattle framework, it is characterized by several parameters such as type of terrain, height, etc. Only one object can be in a cell at a time. Some types of cells can be impassable. 145, 183

Combatant In the vBattle framework, he is an independent agent acting on the environment. He represents a warrior from a given Faction. The Combatants have different characteristics and actions. He must obey the different orders that the Faction Leader issues. 145, 147, 150, 183

Commander A Combatant that is playing the Commander role in the vBattle framework, and acts as relay of the Faction Leader's orders. The Combatants receive the orders from the Faction Leader through the Commander. 145, 183

CP Character Profile. 88, 91, 95, 96, 183

Current Instant Counter In vBattle, the simulated time of the game is divided in Instants (time ticks). The current instant is referred by this counter. 145, 149, 150, 156, 183

Deploy Region A set of cells where a Faction can deploy its Combatants at the beginning of the battle in the vBattle framework. 146, 150, 183

EB Event Builder. 90, 91, 108, 121, 183

EEP Emotional Elicitation Process. 88, 183

EEPE Emotional Elicitation Process Engine. 88, 90, 91, 183

Emotion Based on the Lazarus's Theory of Emotion. The emotions are the psychological response that it is the consequence of the thought produced about the perception of certain events. Emotions are isolated affective responses which accumulate certain tendencies in the affective state of a person. 82, 183

Emotional Event Event produced in an environment that is suitable for the production of emotions. The character processes the perceived events producing the respective emotions. 82, 183

Escape Zone A set of cells, usually on the edges of the Battlefield, that a Combatant must reach so he can flee from the battle. 146, 150, 156, 183

Event vBattle tokens that represent the instant when an action is executed. 145, 183

Event Sequence The ordered sequence of events to simulated in vBattle. 145, 149–151, 156, 158, 183

Faction In the vBattle framework, it is a set of allied Combatants under the command of a Faction Leader. 145, 146, 150, 183

Faction Leader In the vBattle framework, it represents the high level controller of a faction. It can be an artificial intelligence controller or a human player. 145, 146, 183

Gamer A video game player with high involvement and expertise in video games. Usually called "Hardcore-Player". 2, 183

MMOG Massive Multiplayer On-line Game: A video game genre which represents the games that are played usually by a large number of players connected through the internet. 2, 183, 208

Mood Long lasting affective state. With less intensity but with deeper influence on the memory, strategy and assessment of the person that produces a specific emotion. 82, 98, 183

Mood Space Point A particular point in the Mood Vector Space. They are points in the PAD 3D space, noted as $\mu_i \in \mathcal{M}$. 183, 195

Mood State Specific mood at a given instant. In the EEP model it is noted as $\mu_i \in \mathcal{M}$. 26, 46, 82, 88, 90, 98, 183

Mood Tag A label representing a particular mood of a character. It will identify a region in the Mood Vector Space where we allocate a specific mood. This label can have a semantic representation of the mood state. It is noted as $T_i \in \mathcal{T}$. 82, 90, 105, 177, 183, 214

Mood Vector Space The EEP Model component that provides the numerical representation of the mood state of the character. It is modeled as an algebraic space with the operations needed to aggregate the emotions produced, and to evaluate the mood state transitions. 91, 105, 183

MT Mood Tagger. 90, 91, 98, 183

MVS Mood Vector Space. 88, 90, 91, 97, 99, 102, 103, 120–124, 177, 183, 186, 214

Non-Player Character A character present in a video game scene that is controlled by a set of artificial intelligence techniques, so he can perform autonomous actions in the given environment. 183

NPC Non-Player Character. 113, 116, 183

Strategy It is a probability distribution over the space of actions, $A_i \in A$, for each of the states, $\vec{S_i} \in A$. 55, 183

A.2 Symbols and Variables

AGCBAR Architecture

Symbol	Description
$A_i \in \mathbf{A}$	Actions described in the Actions Dictionary
$B_i \in \mathbf{B}$	Strategies, described as $B_i : \mathbf{A} \times \mathbf{S} \to [0, 1]$
$E_i \in \mathbf{E}$	Event Classes described in the Events Dictionary
$G_i \in \mathbf{G}$	Goals described in the Goals Dictionary
$\overrightarrow{S_i} \in \mathbf{S}$	Environment State description represented in the States Dictionary
$T_i \in \mathbf{T}$	Mood Tags described in the Moods Dictionary

EEP Model

Symbol	Description
$\mathbb{A}$	Emotions elicited by the Actions associated to an Event
$a_\omega \in [-1, 1]$	Appeal of the Object ω
$\alpha \in D^a$	An Action like those described in the Actions Dictionary of the EEP Model.
$\mathbb{C}$	Emotions elicited by the Consequences associated to an Event.
D^c, D^a, D^o, D^r and D^g	Conceptual Dictionaries, respectively Consequences, Actions, Objects, Characters and Groups Dictionary.
$d_\gamma \in [-1, 1]$	Desirability of the Consequence γ
$\mathbb{E}$	Set of emotions used in the EEP Engine. $\mathbb{E} = \mathbb{A} \cup \mathbb{C} \cup \mathbb{T} \cup \mathbb{O}$
$\mathcal{E}$	Augmented Event Classes created from AGCBAR Events $\mathbf{E}$.
$e \in \mathbb{E}$	An emotion like those supported by the EEP Engine
$\varepsilon_i \in \mathcal{E}$	Event instance.
$\gamma \in D^c$	A Consequence like those described in the Consequences Dictionary of the EEP Model.
$m^k \subseteq \mathcal{M}$	The set of reference points μ_i^k that represent the mood T_k
$\mathcal{M}$	Mood Vector Space (MVS). Space where the emotions and moods are represented.
$\mu_i \in \mathcal{M}$	A particular point in $\mathcal{M}$ that represents a Mood State.
$\overrightarrow{\nu} \in \mathcal{M}$	Vector representing an emotion (or a group of emotions) in the MVS.
$\mathbb{O}$	Emotions elicited by the Objects associated to an Event.
$\omega \in D^o$	An Object like those described in the Objects Dictionary of the EEP Model.
$p_\alpha \in [-1, 1]$	Praiseworthiness of the Action α
$\mathbb{T}$	Emotions elicited by the composition of Attribution and Well Being emotions.
$r_\rho \in [-1, 1]$	Relationship with the Character ρ or any of the groups that this character belongs to.
$\rho \in D^r$	A Character of those described in the Characters Dictionary of the EEP Model.

WEREWoLF Algorithm

Symbol	Description
$f(w_i)$	WEREWoLF fitness function for individual w_i
$\Gamma(G_i)$	WEREWoLF Goal Interface
$\Lambda(a_i)$	WEREWoLF Action Interface
P_g	WEREWoLF Population of generation g
$\Sigma(\overrightarrow{S_i})$	WEREWoLF Environment State Interface
$r(s)$	WEREWoLF reward function for the state s
w_i	WEREWoLF Individual representing a Reward Matrix

Appendix B

EEP Model Appendix

B.1 Mood Vector Space: Formal Representation

The Mood Vector Space (MVS) is conceived as a formal structure that can represent in the same space the two major items of the emotional behavior models: the emotions and the emotional state (mood). This structure is conceived to accomplish the following requirements that are important for the mood simulation:

- **R1**: Must represent different moods and emotions in the space. According to the PAD representation, bounded in the range $[-1, 1]$.
- **R2**: Support the addition of emotions with the current mood to represent the influence of all of the perceived events over the mood.
- **R3**: The structure must provide the mechanism to classify the continuous value of the mood into a discrete set of mood tags.
- **R4**: The decay of the current mood along the time, moving it to the default mood, usually extracted by the personality of the agent.

This appendix presents the formalism, based on the PAD frameworks, and the functions and operators that are necessary to implement the psychological precepts of dynamics of emotions, intensity of the stimulus and long term tendency of emotional state, that are important for the emotional models. In addition, this appendix shows the functions necessary for the attenuation, composition, etc. by an example of a particular Mood Vector Space (the Trigonometric Mood Space). Still, other functions that match the constraints of the Mood Vector Space formulation [64, 65] can be used.

First, we show the general elements of the MVS and, along the section, we add different components to the formulation of the MVS to create the complete description of the space and the elements that compose it.

B.1.1 Mood Space

A mood space M is define as an algebraic structure $M = (\mathcal{M}, \oplus)$ where the first element is a subset of 3D real number space $\mathbb{R}^3$ bounded between -1 and 1, $\mathcal{M} = [-1, 1]^3$; and the second element is a binary operation in $\mathcal{M}$.

$$\oplus : \mathcal{M} \times \mathcal{M} \longrightarrow \mathcal{M}$$
$$(\overrightarrow{u}, \overrightarrow{v}) \xrightarrow{\oplus} \overrightarrow{u} \oplus \overrightarrow{v}$$

With the following properties:

$$\forall \overrightarrow{u}, \overrightarrow{v} \in \mathcal{M} : \overrightarrow{u} \oplus \overrightarrow{v} \in \mathcal{M} \tag{B.1}$$
$$\exists \overrightarrow{0} \in \mathcal{M} / \forall \overrightarrow{u} \in \mathcal{M} : \overrightarrow{u} \oplus \overrightarrow{0} = \overrightarrow{u} \tag{B.2}$$
$$\forall \overrightarrow{u}, \overrightarrow{v} \in \mathcal{M} : \overrightarrow{u} \oplus \overrightarrow{v} = \overrightarrow{v} \oplus \overrightarrow{u} \tag{B.3}$$
$$\forall \overrightarrow{u} \in \mathcal{M} / \exists \overrightarrow{-u} \in \mathcal{M} : \overrightarrow{u} \oplus \overrightarrow{-u} = \overrightarrow{0} \tag{B.4}$$
$$\forall \overrightarrow{u}, \overrightarrow{v}, \overrightarrow{w} \in \mathcal{M} : (\overrightarrow{u} \oplus \overrightarrow{v}) \oplus \overrightarrow{w} = \overrightarrow{u} \oplus (\overrightarrow{v} \oplus \overrightarrow{w}) \tag{B.5}$$

The properties are:

- Closure property (eq. B.1): For all pair of elements in the space, the result of the operation is also in the space.
- Additive identity property (eq. B.2): There exists an element in the space, such that for all elements in the space keep unchanged after the operation. We name this element as the identity value $\overrightarrow{0}$.
- Commutativity property (eq. B.3): For all pair of elements in the space the operation can exchange the order of parameters.
- Additive inverse property (eq. B.4): For each element in the space, there exists another element in the space, such that the operation returns the identity value $\overrightarrow{0}$.
- Associativity property (eq. B.5): For all three elements in the space the operation allows any associativity aggregation of any pair of elements.

Considering all these properties, the mood space $M = (\mathcal{M}, \oplus)$ is an abelian group.

B.1.2 Mood Vector Space

If we include the scalar multiplication by a real number with following properties:

$$\forall \overrightarrow{u} \in \mathcal{M} \quad \forall \alpha \in \mathbb{R} : \alpha \overrightarrow{u} \in \mathcal{M} \tag{B.6}$$
$$\forall \overrightarrow{u}, \overrightarrow{v} \in \mathcal{M} \quad \forall \alpha \in \mathbb{R} : \alpha(\overrightarrow{u} \oplus \overrightarrow{v}) = \alpha \overrightarrow{u} \oplus \alpha \overrightarrow{v} \tag{B.7}$$
$$\forall \overrightarrow{u} \in \mathcal{M} \quad \forall \alpha, \beta \in \mathbb{R} : (\alpha + \beta) \overrightarrow{u} = \alpha \overrightarrow{u} \oplus \beta \overrightarrow{u} \tag{B.8}$$
$$\forall \overrightarrow{u} \in \mathcal{M} \quad \forall \alpha, \beta \in \mathbb{R} : (\alpha\beta) \overrightarrow{u} = \alpha(\beta \overrightarrow{u}) \tag{B.9}$$
$$\forall \overrightarrow{u} \in \mathcal{M} : 1 \overrightarrow{u} = \overrightarrow{u} \tag{B.10}$$

The properties are:

- Closure property respect multiplication (eq. B.6): For any elements in the space and any scalar value, the result of the multiplication is also in the space.
- Distributivity respect operation (eq. B.7): Scalar multiplication and the operation show distributivity respect $\oplus$ operation.
- Distributivity respect the addition (eq. B.8): Scalar multiplication and the operation show distributivity respect the addition of real numbers.
- Compatibility of scalar multiplication (eq. B.9): The multiplication of real numbers maintains the compatibility.
- Identity element (eq. B.10): 1 denotes the multiplicative identity in $\mathbb{R}$.

In order to be complete, two more operations are defined:

- Subtraction: Defined as the operation with the inverse element.

$$\begin{aligned} \ominus : \mathcal{M} \times \mathcal{M} &\longrightarrow \mathcal{M} \\ (\overrightarrow{u}, \overrightarrow{v}) &\xrightarrow{\ominus} \overrightarrow{u} \ominus \overrightarrow{v} = \overrightarrow{u} \oplus \overrightarrow{-v} \end{aligned} \tag{B.11}$$

- Division by a (non-zero) scalar: Defined as the multiplication by the multiplicative inverse value:

$$\begin{aligned} / : \mathbb{R} \times \mathcal{M} &\longrightarrow \mathcal{M} \\ (\alpha, \overrightarrow{u}) &\xrightarrow{/} \frac{\overrightarrow{u}}{\alpha} = \frac{1}{\alpha}\overrightarrow{u} \end{aligned} \tag{B.12}$$

With all these properties the set $\mathcal{M}$ with the operation $\oplus$ and the field of real numbers $\mathbb{R}$ and the multiplication operator, represent a vector space. Thus, we have denoted the elements of $\mathcal{M}$ as vectors.

Trigonometric Mood Space

An example of a possible mood space description is the trigonometric mood space $M^T = (\mathcal{M}, \oplus_T)$, in which the operation $\oplus_T$ is defined as:

$$\begin{aligned} \forall \overrightarrow{u} = (u_1, u_2, u_3), \overrightarrow{v} = (v_1, v_2, v_3) \in \mathcal{M} : \\ \overrightarrow{u} \oplus_T \overrightarrow{v} = \overrightarrow{w} = (w_1, w_2, w_3) \\ \forall i \in \{1, 2, 3\} : w_i = \frac{2 \arctan\left(\tan\left(u_i \frac{\pi}{2}\right) + \tan\left(v_i \frac{\pi}{2}\right)\right)}{\pi} \end{aligned} \tag{B.13}$$

The proofs for the Closure, Additive Identity, Commutative, Additive Inverse and Associative properties are included in [64, 65].

B.1.3 Extended Mood Space

A extended mood space M is an algebraic structure $M = (\mathcal{M}, \oplus, \odot, \| \cdot \|)$, where $(\mathcal{M}, \oplus)$ is a mood space, presenting also the properties for being a mood vector space (respect $\oplus$ operator and real number multiplication).

The $\odot$ operator is an inner product operator such as:

$$\odot : \mathcal{M} \times \mathcal{M} \longrightarrow \mathbb{R}$$
$$(\overrightarrow{u}, \overrightarrow{v}) \xrightarrow{\odot} \overrightarrow{u} \odot \overrightarrow{v}$$

The $\| \cdot \|$ operator is a norm if satisfies the following properties:

$$\forall \overrightarrow{u} \in \mathcal{M} : \| \overrightarrow{u} \| > 0 \quad \text{if} \quad \overrightarrow{v} \neq \overrightarrow{0} \tag{B.14}$$
$$\forall \overrightarrow{u} \in \mathcal{M} \forall \alpha \in \mathbb{R} : \| \alpha \overrightarrow{u} \| = |\alpha| \, \| \overrightarrow{u} \| \tag{B.15}$$
$$\forall \overrightarrow{u}, \overrightarrow{v} \in \mathcal{M} : \| \overrightarrow{u} \oplus \overrightarrow{v} \| \leq \| \overrightarrow{u} \| + \| \overrightarrow{v} \| \tag{B.16}$$

The properties are:

- Positive length (eq. B.14): For any elements in the space (except the zero element) the norm is always positive. For the zero element $\overrightarrow{0}$ the value is zero.
- Positive homogeneity (eq. B.15): Scalar multiplication and the norm operator resizes the value of the element.
- Triangle inequality (eq. B.16): That is, taking norms as distances, the distance from point A through B to C is never shorter than going directly from A to C, or the shortest distance between any two points is a straight line.

There is a relationship between the inner product $\odot$ and the norm operator $\| \cdot \|$:

$$\| \overrightarrow{u} \| = \sqrt{\overrightarrow{u} \odot \overrightarrow{u}} \tag{B.17}$$
$$\tag{B.18}$$

If M satisfies all these properties is considered a normed vector space.

Trigonometric Extended Mood Space

The trigonometric mood space $M^T = (\mathcal{M}, \oplus_T)$ can become an extended mood space by the definition of:

- $\odot_T$ inner product operation:

$$\forall \overrightarrow{u} = (u_1, u_2, u_3), \overrightarrow{v} = (v_1, v_2, v_3) \in \mathcal{M} :$$
$$\overrightarrow{u} \odot_T \overrightarrow{v} = \sum_{i \in \{1,2,3\}} \tan\left(u_i \frac{\pi}{2}\right) \tan\left(v_i \frac{\pi}{2}\right) \tag{B.19}$$

- $\| \cdot \|_T$ norm operator:

$$\begin{aligned} &\forall \overrightarrow{u} = (u_1, u_2, u_3) \in \mathcal{M} : \\ \| \overrightarrow{u} \|_T &= \sqrt{\overrightarrow{u} \odot_T \overrightarrow{u}} \quad \text{(applying eq. B.17)} \\ &= \sqrt{\sum_{i \in \{1,2,3\}} \tan\left(u_i \frac{\pi}{2}\right) \tan\left(u_i \frac{\pi}{2}\right)} \\ &\qquad \text{(applying eq. B.19)} \\ &= \sqrt{\sum_{i \in \{1,2,3\}} \tan\left(u_i \frac{\pi}{2}\right)^2} \end{aligned} \tag{B.20}$$

The corresponding proofs are also shown in [64, 65].

B.1.4 Topological Mood Space

The existence of a normed vector space M with the properties presented in Section B.1.3, together with $\odot$ operation and the $\| \cdot \|$ operator properties given by the eq.B.14, eq. B.15, eq. B.16, and eq. B.17; provides the possibility to define a topological field K, based on the element addition $\oplus$ and the scalar multiplication. If M is a vector space over a topological field K, M is a topological vector space.

Indeed, all normed vector spaces are topological vector spaces.

Additionally, if the inner product $\odot$ satisfies the following properties:

$$\forall \overrightarrow{u}, \overrightarrow{v} \in \mathcal{M} : \overrightarrow{u} \odot \overrightarrow{v} = \overrightarrow{v} \odot \overrightarrow{u} \tag{B.21}$$

$$\forall \overrightarrow{u}, \overrightarrow{v} \in \mathcal{M}, \alpha \in \mathbb{R} : (\alpha \overrightarrow{u}) \odot \overrightarrow{v} = \alpha(\overrightarrow{u} \odot \overrightarrow{v}) \tag{B.22}$$

$$\begin{aligned} \forall \overrightarrow{u}, \overrightarrow{v}, \overrightarrow{w} \in \mathcal{M} : (\overrightarrow{u} \oplus \overrightarrow{v}) \odot \overrightarrow{w} = \\ (\overrightarrow{u} \odot \overrightarrow{w}) + (\overrightarrow{v} \odot \overrightarrow{w}) \end{aligned} \tag{B.23}$$

The properties are:

- Symmetry (eq. B.21): For any pair of elements in the space the inner product has a conjugate symmetry. As the field on which the inner product is the real number $\mathbb{R}$ and not the complex numbers $\mathbb{C}$ the conjugate symmetry becomes regular symmetry.
- Linearity in the first argument on both product (eq. B.22) and addition (eq. B.23): Scalar multiplication and $\oplus$ operation show linearity respect the first argument considering the addition and the multiplication in $\mathbb{R}$.

Together with the positive-definiteness according to eq. B.14, all these properties make inner product to be complete over the filed $\mathbb{R}$, thus the topological mood space define with this operation is a Hibert space.

The Trigonometric Extended Mood Space satisfies both Symmetry and Linearity of product (see [64, 65]).

B.1.5 Attenuated Mood Space

A attenuated mood space M is an algebraic structure $M = (\mathcal{M}, \oplus, \odot, \|\cdot\|, A)$, where $(\mathcal{M}, \oplus, \odot, \|\cdot\|)$ is a extended mood space and A is a family of functions indexed by $\mathcal{M}$ denoted as $A = \{a_{\overrightarrow{v}} : \overrightarrow{v} \in \mathcal{M}\} = \{a_{\overrightarrow{v}}\}_{\overrightarrow{v}\in\mathcal{M}}$, for which $\forall\overrightarrow{v} \in \mathcal{M}$ is possible to define an infinite sequence: $\langle\overrightarrow{u_n} : n \in \mathbb{N} \quad \overrightarrow{u_n} \in \mathcal{M}\rangle_{\overrightarrow{v}}$ such as:

$$\begin{aligned}
\forall\overrightarrow{u} \in \mathcal{M} : \text{the sequence} &\langle\overrightarrow{u_0}, \overrightarrow{u_1}, \cdots\rangle_{\overrightarrow{v}} \\
&\text{starts with} \quad \overrightarrow{u_0} = \overrightarrow{u} && \text{(B.24)} \\
&\text{generated by func. } a_{\overrightarrow{v}} \text{ as } \overrightarrow{u_i} = a_{\overrightarrow{v}}(\overrightarrow{u_{i-1}}) && \text{(B.25)} \\
&\lim_{n\to\infty} \overrightarrow{u_n} = \overrightarrow{v} && \text{(B.26)}
\end{aligned}$$

The family of functions A represents a set of functions $\{a_{\overrightarrow{v}}\}_{\overrightarrow{v}}$ that converge to given values of $\overrightarrow{v}$ for all the possible initial elements $\overrightarrow{u_0} \in \mathcal{M}$.

In order to be completed, it is necessary to define how the limit of the sequence is computed (eq. B.26). As M is a normed vector space according to the properties presented in Section B.1.3.

$$\lim_{n\to\infty} \overrightarrow{u_n} = \overrightarrow{v} \tag{B.27}$$

$$\lim_{n\to\infty} \| \overrightarrow{u_n} \ominus \overrightarrow{v} \| = 0 \tag{B.28}$$

Trigonometric Attenuated Mood Space

The trigonometric extended mood space $M^T = (\mathcal{M}, \oplus_T, \odot_T, \|\cdot\|_T)$, can become an attenuated mood space by the definition of a family of function A_T^α indexed by $\mathcal{M}$, such as:

$$\begin{aligned}
A_T^\alpha = \{a_{T,\overrightarrow{v}}^\alpha : \overrightarrow{v} \in \mathcal{M}\} &= \{a_{T,\overrightarrow{v}}^\alpha\}_{\overrightarrow{v}\in\mathcal{M}} \\
\forall\overrightarrow{v} \quad \text{there is a function} \quad a_{T,\overrightarrow{v}}^\alpha \quad &\text{such as:} \\
\forall\overrightarrow{u} : a_{T,\overrightarrow{v}}^\alpha(\overrightarrow{u}) &= \overrightarrow{u} \oplus_T \alpha(\overrightarrow{v} \ominus_T \overrightarrow{u}) && \text{(B.29)}
\end{aligned}$$

For any $\alpha \in (0, 1)$ the structure $M^T = (\mathcal{M}, \oplus_T, \odot_T, \|\cdot\|_T, A_T^\alpha)$ is an attenuated mood space.

The Trigonometric Attenuated Mood Space satisfies these properties (see [64, 65]).

B.1.6 Emotional Agent System in a Mood Vector Space

It is possible to define an Emotional Agent System $\mathcal{A}$ as an algebraic structure $\mathcal{A} = (M, A, E, m_0)$, where M is a Mood Vector Space, A a finite set of agents $A = \{A_0, A_1, \cdots, A_n\}$, and E is a set of elements defined as $E = \{(a, t, v, \alpha)/a \in A, t \in$

$\mathbb{N}, v \in \mathcal{M}, \alpha \in \mathbb{R}\}$, named as the emotion set, which represents all the emotions elicited by all the agents, together with its intensity at a give time step. Finally, m_0 is a function that represents the default mood state for the agents (mood state in absence of any emotion, thus the initial state):

$$m_0 : A \longrightarrow \mathcal{M} \quad : \quad A_i \xrightarrow{m_0} \overrightarrow{u}_i^0 \tag{B.30}$$

The state of an Emotional Agent System, can be represented as the mood state of all its agents, and it is denoted as $M(\mathcal{A}, t) = \{\overrightarrow{u}_0^t, \overrightarrow{u}_1^t, \cdots, \overrightarrow{u}_n^t\}$, being $t \in \mathbb{N}$. This state can be defined as follows:

$$\forall i \in [0, n] : \text{if } t = 0 \quad \overrightarrow{u}_i^0 = m_0(A_i) \quad \text{Initial mood state} \tag{B.31}$$

$$\text{if } t > 0 \quad \overrightarrow{u}_i^t = \overrightarrow{u}_i^{t-1} \oplus E_{(A_i,t)} \tag{B.32}$$

$$\text{where } E_{(A_i,t)} = \alpha_0 \overrightarrow{v_0} \oplus \cdots \oplus \alpha_m \overrightarrow{v_m}$$

$$\forall (A_i, t, v_j, \alpha_j) \in E \tag{B.33}$$

$E_{(A_i,t)}$ represents the aggregation of all the emotions elicited by the agent A_i at time stamp t scaled according to the intensities α_j and combined by means of the $\oplus$ operator.

If we want to include a mechanism that ensures that the mood state of the agents returns to their default initial state along time (and in absence of new emotions); we must satisfy that M is an Attenuated Mood Space in order to redefine the equation B.32 as:

$$\text{if } t > 0 \quad \overrightarrow{u}_i^t = a_{\overrightarrow{v}}(\overrightarrow{u}_i^{t-1} \oplus E_{(A_i,t)}) \tag{B.34}$$

where $\overrightarrow{v} = m_0(A_i)$ is the default (initial) mood state and $a_{\overrightarrow{v}}(\cdot)$ is the function, from the family of function A in the Attenuated Mood Space, that defines the infinite sequence that converges to this default (initial) mood state $m_0(A_i)$.

Also, we can include a mechanism to discretize the MVS in order to obtain a specific mood tag $M_i^\tau \in M^\tau$. To do it so we include a function of neighborhood $f_\nu(A_i)$ (for instance a minimal norm distance function) that returns the mood tag from the current mood for the specific agent A_i:

$$f_\nu : \mathcal{M} \longrightarrow \mathcal{M}^\tau \quad : \quad \overrightarrow{u}_i^t \xrightarrow{f_\nu} M_i^\tau \tag{B.35}$$

An Emotional Agent System supports the requirements proposed as objective for this representation of emotions and moods.

B.2 Mood Ontology

In the EEP Model, we describe a possible mood state ontology, shown in the Figure B.1, to represent the different mood states of the characters. This ontology is based on the PAD quadrants, for each of the dimensions and sign ($\pm P$, $\pm A$, $\pm D$). This

representation is proposed for two usages: first, we use the tags that can describe the different mood state tags ($T \in \mathbf{T}$) to include them in the Architectural Dictionaries of the AGCBAR Architecture and to use as output of the EEP Model, and, second we describe the different Mood Space Points according to the representation of those points so they were near of the desired tags. These two usages are based on the Russell & Mehrabian's works[49, 74] which summarize the values that are representative for a set of temperament tags.

The root of the ontology is the "zero-informative" mood (in the figure it is tagged as `Mood`). The first level of the ontology is the six different values of the axis of the PAD space, each of the encloses four of the quadrants of the space (i.e. `Pleasant`$\equiv \pm P$ describe those mood states that has positive pleasure, like `Relaxed` or `Exuberant`). The second level of the hierarchy represent a disjoint union of the eight quadrants that is proposed as basis in the Mehrabian's work. Thus, we have the eight different tags for the PAD Octants, one for each of these subspaces (`Disdainful`, `Relaxed`, etc.).

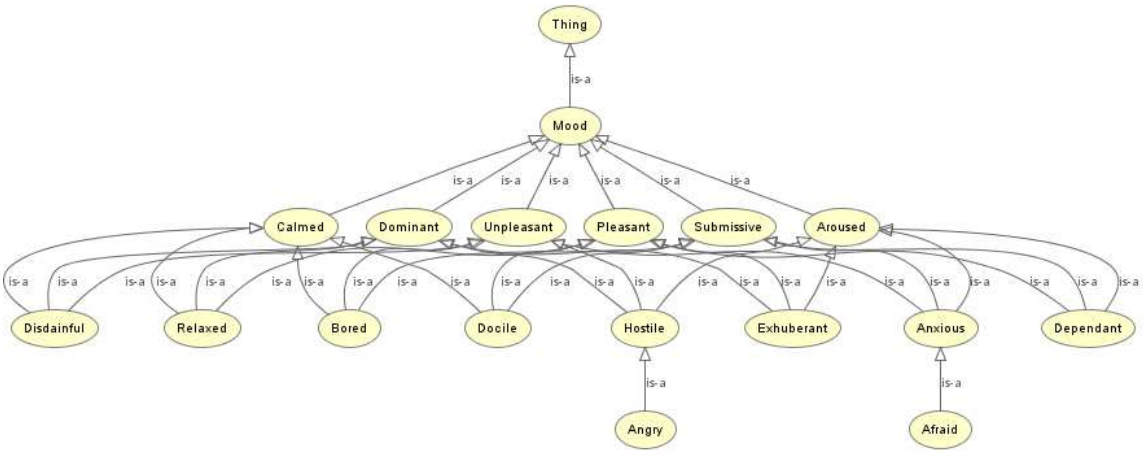


Figure B.1: EEP Mood Ontology Example. Based on the PAD Temperament Model[49]. This ontology presents a possible division of moods to ensure the coverage of all of the possible moods in order to create the strategies.

The different levels of the hierarchy represent more specialization of the mood, with more level of detail in the kind of mood that it represents. For instance, `Afraid` is a kind of mood that is `Unpleasant`, `Submissive` and `Aroused`, and also more specific than `Anxious`. See Figure B.2.

With this kind of representation we can determinate the moods that we want to use in a specific scenario for the based on those mood state supported by the EEP Model. The characters can identify the moods relevant for the desired behaviors from those proposed and ensure the completeness of the mood states handled by the character.

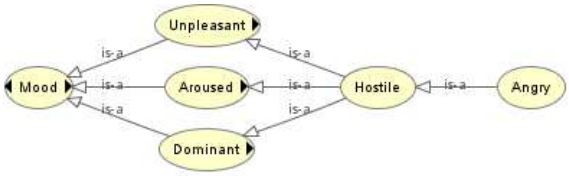


Figure B.2: EEP Mood Ontology Example. Angry is a specific mood which is somehow `Hostile` which is an `Unpleasant`, `Aroused` and `Dominated` mood state.

B.3 Combat Scenario Example

B.3.1 Orc Boss Profile

```
<?xml version="1.0" encoding="UTF-8"?>
<eep:profile xmlns:eep="http://www.ia.urjc.es/eep/ProfileSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ia.urjc.es/eep/ProfileSchema ../xsd/ProfileSchema.xsd ">
  <eep:consequences>
    <eep:consequence eep:desirability="0.6" eep:tag="HITENEMY" />
    <eep:consequence eep:desirability="-0.2" eep:tag="BEHIT" />
    <eep:consequence eep:desirability="0.7" eep:tag="KILLENEMY" />
    <eep:consequence eep:desirability="-0.2" eep:tag="BEKILLED" />
    <eep:consequence eep:desirability="-0.1" eep:tag="BEATTACKED" />
    <eep:consequence eep:desirability="-0.3" eep:tag="MOVEDAWAY" />

  </eep:consequences>
  <eep:actions>
    <eep:action eep:praiseworthiness="0.5" eep:tag="ATTACK" />
    <eep:action eep:praiseworthiness="-0.3" eep:tag="DEFEND" />
    <eep:action eep:praiseworthiness="-0.7" eep:tag="MOVEAWAY" />
  </eep:actions>
  <eep:objects>
    <eep:object eep:appealing="0.0" eep:tag="HUMANBOSS" />
    <eep:object eep:appealing="0.1" eep:tag="ORCBOSS" />
    <eep:object eep:appealing="0.0" eep:tag="ORC" />
  </eep:objects>
  <eep:relations>
    <!--eep:relation eep:friendship="0.2" eep:tag="OrcBoss" /-->
    <eep:relation eep:friendship="0.2" eep:tag="OrcSoldier" />
    <eep:relation eep:friendship="0.2" eep:tag="OrcArcher" />
    <eep:relation eep:friendship="-0.5" eep:tag="HumanBoss" />
    <eep:relation eep:friendship="-0.5" eep:tag="HumanSoldier" />
    <eep:relation eep:friendship="-0.5" eep:tag="HumanArcher" />
  </eep:relations>
  <eep:personality eep:agreeableness="-0.5" eep:conscientiousness="-0.5"
  eep:extraversion="0.1" eep:neuroticism="0.3" eep:openness="-0.4" />
  <eep:moodMap>
   <eep:mood eep:tag="NORMAL">

   <eep:location eep:pleasure="0.1" eep:arousal="0.2" eep:dominance="0.3" />

   </eep:mood>
   <eep:mood eep:tag="AFRAID">

   <eep:location eep:pleasure="-0.8" eep:arousal="0.8" eep:dominance="-0.8" />
```

```
   </eep:mood>
   <eep:mood eep:tag="ANGRY">

   <eep:location eep:pleasure="-0.1" eep:arousal="0.6" eep:dominance="0.6" />

   </eep:mood>
</eep:moodMap>

</eep:profile>
```

B.3.2 Orc Boss's Mood Evolution Through Time Steps

Time	State	P	A	D	$d(\mu, \text{NORMAL})$	$d(\mu, \text{ANGRY})$	$d(\mu, \text{AFRAID})$
0	NORMAL	-0,22	-0,04	0,04	0,48*	0,86	1,32
1	NORMAL	0,56	0,56	0,51	0,62*	0,67	1,90
2	ANGRY	0,76	0,74	0,67	0,93	0,88*	2,15
3	ANGRY	0,85	0,82	0,77	1,08	0,99*	2,27
4	ANGRY	0,88	0,86	0,80	1,14	1,03*	2,32
5	ANGRY	0,90	0,88	0,83	1,18	1,06*	2,36
6	ANGRY	0,91	0,90	0,86	1,21	1,08*	2,38
7	ANGRY	0,92	0,90	0,87	1,22	1,10*	2,40
8	ANGRY	0,93	0,92	0,89	1,25	1,12*	2,42
9	ANGRY	0,93	0,92	0,89	1,25	1,12*	2,42

Table B.1: Combat Scenario: Mood State Evolution of the Orc Boss.

B.3.3 Human Archer Profile

```
<?xml version="1.0" encoding="UTF-8"?>
<eep:profile xmlns:eep="http://www.ia.urjc.es/eep/ProfileSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.ia.urjc.es/eep/ProfileSchema ../xsd/ProfileSchema.xsd ">
  <eep:consequences>
    <eep:consequence eep:desirability="0.1" eep:tag="HITENEMY" />
    <eep:consequence eep:desirability="-0.6" eep:tag="BEHIT" />
    <eep:consequence eep:desirability="0.4" eep:tag="KILLENEMY" />
    <eep:consequence eep:desirability="-0.9" eep:tag="BEKILLED" />
    <eep:consequence eep:desirability="-0.05" eep:tag="BEATTACKED" />
    <eep:consequence eep:desirability="-0.2" eep:tag="MOVEDAWAY" />

  </eep:consequences>
  <eep:actions>
    <eep:action eep:praiseworthiness="0.04" eep:tag="ATTACK" />
    <eep:action eep:praiseworthiness="0.2" eep:tag="DEFEND" />
    <eep:action eep:praiseworthiness="-0.4" eep:tag="MOVEAWAY" />
```

```
  </eep:actions>
  <eep:objects>
    <eep:object eep:appealing="0.6" eep:tag="HUMANBOSS" />
    <eep:object eep:appealing="-0.1" eep:tag="ORCBOSS" />
    <eep:object eep:appealing="0.0" eep:tag="ORC" />
  </eep:objects>
  <eep:relations>
    <eep:relation eep:friendship="1.0" eep:tag="HumanBoss" />
    <eep:relation eep:friendship="0.8" eep:tag="HumanSoldier" />
    <!--eep:relation eep:friendship="0.9" eep:tag="HumanArcher" /-->
    <eep:relation eep:friendship="-0.3" eep:tag="OrcArcher" />
    <eep:relation eep:friendship="-0.3" eep:tag="OrcBoss" />
    <eep:relation eep:friendship="-0.3" eep:tag="OrcSoldier" />
  </eep:relations>
  <eep:personality eep:agreeableness="0.5" eep:conscientiousness="-0.1"
  eep:extraversion="0.4" eep:neuroticism="-0.1" eep:openness="0.4" />
  <eep:moodMap>
   <eep:mood eep:tag="NORMAL">

   <eep:location eep:pleasure="0.2" eep:arousal="-0.1" eep:dominance="0.3" />

   </eep:mood>
   <eep:mood eep:tag="AFRAID">

   <eep:location eep:pleasure="-0.2" eep:arousal="0.6" eep:dominance="-0.7" />

   </eep:mood>
   <eep:mood eep:tag="ANGRY">

   <eep:location eep:pleasure="-0.2" eep:arousal="0.8" eep:dominance="0.4" />
   <eep:location eep:pleasure="-0.3" eep:arousal="0.5" eep:dominance="0.1" />
   </eep:mood>


</eep:moodMap>

</eep:profile>
```

B.3.4 Combat Scenario Questionnaire

Question	**Rating**[1]
(1) Without considering the underneath affections and liking/disliking relationships among the characters:	
(1.a) Do you consider the behavior of the *Orc Boss* as believable?	1 2 3 4 5
(1.b) Do you consider the behavior of the *Human Archer* as believable?	1 2 3 4 5
(2) Using the information about the character profile (*Orc Boss* is short-tempered and *Human Archer* is fearful)	
(2.a) Do you consider the behavior of the *Orc Boss* as believable?	1 2 3 4 5
(2.b) Do you consider the behavior of the *Human Archer* as believable?	1 2 3 4 5
(3) Using the information about the effect that some consequences have in the characters' mood (the *Orc Boss* is enraged when it is hit by a human enemy and the *Knight* killed by the orcs is considered by the archer as been doomed)	
(3.a) Do you consider the behavior of the *Orc Boss* as believable?	1 2 3 4 5
(3.b) Do you consider the behavior of the *Human Archer* as believable?	1 2 3 4 5
(4) As a summary (and knowing all the information), does the overall sequence shown in the example seem believable?	1 2 3 4 5

Table B.2: Combat Scenario: Evaluation Questionnaire.

B.4 Storytelling Scenario Example

B.4.1 Sad Girl Profile

```
<?xml version="1.0" encoding="UTF-8"?>
<eep:profile xmlns:eep="http://www.ia.urjc.es/eep/ProfileSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ia.urjc.es/eep/ProfileSchema ../xsd/ProfileSchema.xsd ">
  <eep:consequences>
    <eep:consequence eep:desirability="-0.5" eep:tag="Talk-RUD" />
    <eep:consequence eep:desirability="0.6" eep:tag="Talk-POL" />
    <eep:consequence eep:desirability="0.1" eep:tag="Talk-NEU" />
    <eep:consequence eep:desirability="0.9" eep:tag="Quest-Cat-Done" />

  </eep:consequences>
  <eep:actions>
    <eep:action eep:praiseworthiness="0.1" eep:tag="Talk" />
    <eep:action eep:praiseworthiness="-0.5" eep:tag="Steal" />

  </eep:actions>
  <eep:objects>
    <eep:object eep:appealing="0.4" eep:tag="Song-Happy" />
    <eep:object eep:appealing="0.5" eep:tag="Cat" />
  </eep:objects>
  <eep:relations>
```

```
    <eep:relation eep:friendship="0.5" eep:tag="Player" />
    <eep:relation eep:friendship="0.7" eep:tag="BreardedMerchant" />
    <eep:relation eep:friendship="0.3" eep:tag="MelancholicBard" />
  </eep:relations>

  <eep:personality eep:agreeableness="0.5" eep:conscientiousness="-0.1"
  eep:extraversion="0.4" eep:neuroticism="-0.1" eep:openness="0.4" />
  <eep:moodMap>
   <eep:mood eep:tag="NORMAL">

   <eep:location eep:pleasure="0.2" eep:arousal="-0.1" eep:dominance="0.3" />

   </eep:mood>
   <eep:mood eep:tag="HAPPY">

   <eep:location eep:pleasure="-0.2" eep:arousal="0.6" eep:dominance="-0.7" />

   </eep:mood>
   <eep:mood eep:tag="SAD">

   <eep:location eep:pleasure="-0.2" eep:arousal="0.8" eep:dominance="0.4" />

   </eep:mood>

</eep:moodMap>

</eep:profile>
```

B.4.2 Storytelling Questionnaire

(1) Provide the sequence of actions followed to solve the scenario:

(__) Speak POLITE with the *Sad Girl*
(__) Speak RUDE with the *Sad Girl*
(__) Speak NEUTRAL with the *Sad Girl*
(__) Speak POLITE with the *Bearded Merchant*
(__) Speak RUDE with the *Bearded Merchant*
(__) Speak NEUTRAL with the *Bearded Merchant*
(__) Buy something to the *Bearded Merchant*
(__) Speak POLITE with the *Skinny Merchant*
(__) Speak RUDE with the *Skinny Merchant*
(__) Speak NEUTRAL with the *Skinny Merchant*
(__) Buy something to the *Skinny Merchant*
(__) Obtain a rope

Question	**Rating**[2]
(2) After analyzing the actions performed and the corresponding results:	
(2.a) Do you consider the behavior of the *Sad Girl* as believable?	1 2 3 4 5
(2.b) Do you consider the behavior of the *Bearded Merchant* as believable?	1 2 3 4 5
(2.c) Do you consider the behavior of the *Melancholic Bard* as believable?	1 2 3 4 5
(3) As a summary (and knowing all the information), does the overall character interaction seem believable?	1 2 3 4 5

Table B.3: Storytelling Scenario: Evaluation Questionnaire

Appendix C

WereWoLF Algorithm Appendix

Combatant Example

Anacletus			HPs	320	EP	150
Attack	APs	EPs	Dam HPs	Dam EPs	Stun	% Hit
AnHard	15	50	75	45	0	0.55
AnQuick	5	15	10	0	0	0.70
AnStun	10	30	10	0	20	0.65
AnBalanced	10	30	20	0	0	0.50
AnDummy	20	80	5	0	0	0.20
Defense	APs	EPs	Dam Red	Stun Avoid	% Block	
AnReduction	10	30	0.20	true	0.50	
AnLong	30	30	0.60	false	0.50	
AnEffective	10	15	0.70	true	0.90	

Where *HPs* are the Hit Points (the health of a combatant), the *EPs* are the Exhaustion Points of the combatant (the energy of a combatant), and the *APs* are the Action Points associated to an action (the time consumed to perform an action).

Simplified vBattle Engine

Algorithm 4: Simplified vBattle Engine

begin
Let B, R be two combatants with $A_B = \{A_{B_1}, \ldots, A_{B_n}\}, A_R = \{A_{R_1}, \ldots, A_{R_m}\} \in A$ the sets of actions for B and R.
while *Finalization state not reached* **do**
Let $seg_i = \{A_j, \ldots\}$ next segment of Actions and $ini \in \mathbb{N}$ the next instant counter
RecoverEnergy(B, R, ini)
RecoverStun(B, R, ini)
for A_j **do**
if A_j *is Attack* **then**
Resolve$(A_j, S, T) \quad \rightarrow D_j \in \bar{D}$
end
EnergyModification(A_j, S)
end
ApplyDamages$(\bar{D})$
if B *or* R *has no action declared* **then**
DeclareAction$(c \in \bar{C})$
end
end
end
Where the function $RecoverEnergy(B, R, ini)$ adds an amount of energy to the combatant; and $RecoverStun(B, R, ini)$ updates the counter of time remaining to finish the time while a combatant is stunned.
Resolve$(A_j, S, T) \quad \rightarrow D_j$
begin
if A_j *hits* **then**
HitGrade$(A_j) \rightarrow G_{off}$
if T *has a defence declared* **then**
DefenceGrade$(A_{T_i}) \rightarrow G_{def}$
end
DamageCalculus$(G_{off} - G_{def}, A_{T_i}) \rightarrow D_j$
end
else
$0 \rightarrow D_j$
end
return D_j
end

Appendix D

vBattle Experimentation Framework

D.1 AGCBAR Architectural Dictionaries

Event Dictionary

The following events can produce changes in the characters' mood state in the vBattle experiment:

- `Move(`*`Combatant`*`,`*`Position`*`)`: Produced each time a combatant completes a movement.
- `Attack(`*`CombatantSrc`*`,`*`CombatantTgt`*`,`*`Result`*`)`: Every attack made can produce an emotional response in the characters that perceive it.
- `Wound(`*`Combatant`*`,`*`Wound`*`)`: The wounds produced by an attack.
- `Dead(`*`Combatant`*`)`: A Combatant's death is an important event to the mood state evolution.
- `Fled(`*`Combatant`*`)`: In many terms, this event is as important as the `Dead` event.

Moods Dictionary

The *Mood Dynamics* adopted in the vBattle relies on the following moods:

- `Normal`: The default mood for the combatants. The orders are followed and the combatants behave according to the objectives of the scenario.
- `Angry`: It is an uneasy state where the combatant tends to act more offensively.
- `Afraid`: In this state, the combatants become more defensive because the adverse events they perceived makes them be concerned about their own safety.

- `Broken`: This state represents a total loss of courage. The Broken characters should try to flee desperately.

Environment State Dictionary

The representation of the environment in the vBattle is made according to the complete visual information within the range of the characters. The characters, in absence of vision constrains such as "fog-of-war" (a vision range limitation), have a complete information of the environment. This information includes: the Battlefield scenario elements, i.e. the cell information, the combatants and their states; and is also includes the rest of the game state information.

Actions Dictionary

The actions described in this dictionary are the combatant's actions specified in the Core Rules of the vBattle. We describe the actions of moving to each of the six possible positions in the neighborhood of a combatant. The different attacks of the combatant are described, from the low-level actions, as:

- `Soft Attack`: The default attack. Neither the fastest nor the most damaging.
- `Quick Attack`: The fastest attack launched by a combatant, in terms of the APs consumed.
- `Hard Attack`: The most damaging attack of the combatant.
- `Dummy Attack`: An attack that is not associated to any of the previous ones.

The defenses are described as in the vBattle design:

- *No-Defense*: The Combatant will not try to defend himself from the incoming attack
- *Soft Block*: A less time-consuming defensive action.
- *Block*: The default defense of the combatant when he is in melee combat.
- *Full Block*: An improved but very time-consuming defensive action.
- *Dodge*: This is a special type of defense applicable while the combatant is running or firing a projectile.

The combatant also has two additional actions:

- `Run`: Action to increase the movement rate,
- `Rest`: Action to recover the Fatigue Points.

Goals Dictionary

Given the different rules of the vBattle, we describe the basic goals of:

- `Preserve Energy`: make efficient attacks and defenses and do not run.
- `Maximize Enemy Damage`: when we try to find the way to kill or damage as many enemies as we can.
- `Minimize Own Damage`: the objective is to remain alive and keep fighting.
- `Run Away`: the combatant tries to reach the closest Escape Zone in the minimum time.

D.2 EEP Model Conceptual Dictionaries

Consequences	
HitEnemy	BeHit
KillEnemy	BeKilled
BeAttacked	Fled
WoundEnemy	BeWounded
MissAttack	BlockAttack
Inferiority	Superiority
Actions	
Attack	Block
Flee	
Objects	
Inferiority	Superiority

Appendix E

Resumen en español

–¿Y tu papá en qué trabaja?
– No sé, juega al ordenador.

Caludia Peña

E.1 Antecedentes

E.1.1 ¿Por qué hacer una tesis sobre videojuegos?

La principal razó para investigar en el campo de los videojuegos es la perspectiva de negocio, debida a la constante expansión económica the este sector. [1]

El último informe sobre las tendencias de mercado realizado por "Gaming Ecosystems 2011" [2] constata que en EE.UU. la venta de hardware y juegos superó en 2010 los 67,000 millones de dólares y en 2011 los 74,000 millones de dólares. Estos datos indican que la industria del videojuego se posiciona a la cabeza del consumo de entretenimiento y ocio, superando al cine y la música desde los principios del siglo XXI, así se muestra en el análisis de CBS: el consumo de videojuegos en EE.UU. con 23,000 millones de dólares es tres veces mayor que el gasto en música, 6,900 millones, y el doble que el cine, 10,600 millones. En cuanto al desarrollo de software, los números indican que el desarrollo de videojuegos no sólo representa la rama más rentable de Ingeniería y Telecomunicaciones, sino que es la que mayores ingresos brutos genera.

El desarrollo de videojuegos, como cualquier otra industria dedicada al desarrollo de software, tiene sus propias particularidades: se circunscribe a un ciclo de desarrollo muy corto en el que las actividades de los distintos grupos, como diseñadores gráficos y músicos,

[1] http://www.theesa.com/facts/pdfs/VideoGames21stCentury_2010.pdf

[2] http://www.gartner.com/newsroom/id/1737414

guionistas o editores, consumen un volumen significativo de tiempo y de recursos, comparado con los programadores [8]. Con este panorama, el desarrollo de software puro queda restringido a determinadas áreas: inteligencia artificial, programación de efectos gráficos y unificación de las diferentes partes del proyecto. Este tipo de proceso de desarrollo se basa, fundamentalmente, en potentes entornos de trabajo capaces de soportar un desarrollo rápido y eficaz [70] que incluyen: motores gráficos, mecanismos para el control de eventos, plantillas para los mecanismos de control, algoritmos de búsqueda, planificación y toma de decisiones, entre otros.

Hoy en día, la evolución de la industria de los videojuegos se encamina hacia la consecución de productos cada vez más realistas, no sólo en términos de la calidad visual o gráfica, sino también en el refinamiento de los patrones de conducta y comportamiento de los diferentes protagonistas del juego.

En los últimos años, los juegos multijugador online (MMOG), como *World of Warcraft*, aparecen para satisfacer las necesidades de los usuarios más exigentes que buscan en los personajes algo más que un perfil estereotipado, este plus lo aportan otros jugadores humanos quienes se convierten en un enemigo o un aliado interesante y que proporcionan realismo, reaccionando de manera verosimil y sorprendente antes los distintos acontecimientos que se van sucediendo durante el transcurrir del juego; este tipo de conductas no se ha logrado simular plenamente en los personajes de los jugadores virtuales cuyos patrones de comportamiento se basan en controladores de inteligencia artificial. Los MMOG, por tanto, suponen una revolución que aporta valor añadido enriqueciendo la experiencia de usuario y produciendo el disfrute de un gran número de jugadores que interactúan en el mismo espacio tiempo virtual.

Los MMOG son una realidad que ha conseguido, en los últimos años, impulsar la industria de los videojuegos, así lo constata el informe de tendencias Newzoo [3], el impacto de este tipo de juegos es cada vez mayor en número de jugadores, porcentaje de usuarios dispuestos a pagar por jugar y el beneficio económico generado. Sin embargo, los jugadores demandan juegos cada vez más sofisticados, no sólo en la consecución de entornos gráficos más realistas, sino en otros muchos aspectos como: patrones de comportamiento humano, estrategias de adaptación de los oponentes, fluidez y nivel de generación de escenarios o la interacción con agentes externos y el medio ambiente.

Además, la industria del videojuego, en particular la industria europea, tiene que competir con grandes compañías de EE.UU. o con empresas capaces de reclutar grandes equipos para el diseño y desarrollo de sucesivas versiones de cualquiera de los "juegos de clase AAA" (producciones de videojuego con gran presupuesto) que cada año ven la luz en todo el mundo. Estos equipos pueden desarrollar no sólo el contenido para el nuevo juego, sino también recursos adicionales para actualizar o volver a diseñar la plataforma de software que utilizan.

El éxito del desarrollo de videojuegos, hoy por hoy, no sólo se debe a los avances

[3] http://www.newzoo.com/trend-reports/mmo-trend-report/

tecnológicos que le acompañan, también tiene un papel importantísimo la creatividad de los diseñadores de juegos en la concepción o el enfoque de cada nuevo producto. [4] Un claro ejemplo, lo encontramos en las tres principales consolas de uso doméstico: Sony PS3 ha sido la más avanzada en cuanto a tecnología base, seguida de Microsoft Xbox y la zaga de ambas tendríamos Nintendo Wii. A pesar de la ventaja tecnológica de Sony, los desarrolladores Wii han hecho sobresalir a Nintendo gracias al éxito alcanzado por el diseño de juegos altamente participativos e interactivos que se han abierto a nuevos mercados, hecho que se ha visto reflejado en el aumento de la cuota de mercado en las acciones de Nintendo consiguiendo un 27 % más que las acciones de Sony. Este hecho no hubiera sido posible, si todo esto no hubiera ido acompañado de la aparición de nuevos dispositivos capaces de hacer llegar al usuario esas brillantes ideas de juegos interactivos.

Todo lo dicho anteriormente, y observando las cifras económicas que se manejan entorno a la industria del videojuego, queda patente su importancia en términos de inversiones, así como el interés de los distintos campos de investigación de las ciencias computacionales que se pueden ver implicadas en este tipo de proyectos[5].

¿Por qué escribir una tesis de IA de videojuegos?

En primer lugar, la necesidad de desarrollar tecnologías que se adapten a las necesidades de la industria del videojuego, tal y como mencionaba Ralph Edwards en su artículo sobre la economía de juegos[6]:

> *. . . el tamaño de los equipos necesarios, para hacer juegos para las consolas más modernas, duplica, en comparación con la generación anterior, el número de modeladores, animadores y otros artistas necesarios, por lo qué el coste de desarrollo aumenta significativamente para cada nueva generación de consolas.*

Por otra parte, los "juegos de clase AAA" requieren modelos de personajes cada vez más sofisticados. Sin embargo, mientras que en este tipo de juegos los aspectos visuales de los personajes virtuales están, por lo general, bien logrados, recientemente su comportamiento ha pasado a considerarse tan importante como los efectos visuales. En los últimos años, el desarrollo de juegos ha aplicado soluciones genéricas estáticas que derivaron en modelos de comportamiento nada flexibles y poco realistas; esto puede provocar que los jugadores se sientan poco satisfechos con el juego ya que quieren es percibir la sensación de que los personajes en el escenario se mueven impulsados por un propósito, una meta, y no que parezca que sólo están deambulando por la escena. Con el fin de conseguir esto, se han adaptado técnicas de razonamiento para producir en los personajes virtuales un comportamiento "racional", así como en las interacciones que el jugador tenga con ellos.

[4] http://www.vgchartz.com/tools/hw_date.php?Reg=Global&=anual_term

[5] http://www.polygon.com/2012/10/1/3439738/the-state-of-games-state-of-aaa

[6] http://www.ign.com/articles/2006/05/06/the-economics-of-game-publishing

Para lograr credibilidad y realismo en los personajes virtuales, los juegos deben ser también capaces de, a veces, sorprender y desafiar al jugador con la toma de decisiones no del todo racionales, motivadas por una respuesta emocional ante el entorno y los acontecimientos recientes. Esta tarea crece en complejidad, con el número de diferentes agentes de juego o con el aumento del número de acciones posibles.

Aun cuando la calidad gráfica de un juego es uno de los principales atractivos para la venta, la fidelidad de los usuarios, la compra de nuevas ediciones, año tras año, se basa en otros aspectos como el guión o la historia. Un factor clave en esta experiencia de juego es el comportamiento de los agentes en el juego [33]. Debemos tener en cuenta que el comportamiento de los agentes, así como el resto de los aspectos del juego, debe responder a las exigencias de esta industria. Teniendo en cuenta todos estos aspectos, el ajuste en costes es un requisito del diseño de controladores de agente de los videojuegos modernos.

Como se ha indicado anteriormente, además del componente gráfico, el éxito de un juego también depende, en gran medida, de otros aspectos como la trama o la experiencia de juego. Un factor clave de la experiencia de juego es el comportamiento de los agentes, que son controlados por las rutinas de inteligencia artificial (controladores de agente y planificadores) [91]. La inteligencia artificial en los videojuegos ha evolucionado mucho en los últimos años, técnicas de inteligencia más avanzadas han sustituido a los controladores basados en reglas tradicionales. A pesar de que estos nuevos métodos han mejorado la calidad de los nuevos controladores, requieren una gran cantidad de codificación manual y ajuste lo que supone un gran esfuerzo de los desarrolladores. Por otra parte, el desarrollo de comportamientos más creíbles y semejantes a los humanos sigue siendo una asignatura pendiente en los videojuegos modernos.

La industria tiene que adoptar nuevas tecnologías para lograr estos resultados en un marco limitado de tiempo, consecuencia de los tiempos y los flujos de trabajo implicados en su desarrollo [17]. Por desgracia, la industria, empujada por los plazos del mercado, invierte menos de lo que le gustaría a la investigación para producir nuevos avances en este campo. La razón es que es importante para acelerar el tiempo de desarrollo para controladores de juego para alcanzar la mejor calidad de el controlador en una cantidad limitada de tiempo. Tienden a aplicar esos procedimientos bien establecidos para desarrollar controladores existentes.

La industria sigue toda la trayectoria anteriormente mencionada, en la que encuentra cabida la investigación activa en mecanismos de Inteligencia Artificial adaptativa y personalizada con dos objetivos fundamentales: **(1) modelado de comportamiento de personajes, para lograr conductas más creíbles y (2) colaborar en el de diseño de juegos con la codificación de dispositivos de juego inteligentes.**

E.2 Objetivos

La motivación de esta tesis está fundamentada en la idea: "El éxito de un nuevo videojuego no sólo se basan en la tecnología, también está influenciado por el conjunto de ideas que contribuyen a la creación de un entorno desafiante, adictivo y novedoso".

Lo que conduce a la siguiente hipótesis:

> *Se puede mejorar el desarrollo de los controladores de los personajes virtuales de los videojuegos aplicando técnicas de aprendizaje y rasgos de personalidad.*

La creación automática, o asistida por ordenador, de los controladores, unida a la aplicación de modelos emocionales en el desarrollo de caracteres de personajes, puede contribuir mejorando el diseño de entornos simulados en los videojuegos. Se pueden incluir en el ciclo de desarrollo de videojuegos herramientas de apoyo, que doten de mayor realidad al entorno y a la experiencia del jugador, construyendo personajes virtuales más creíbles. Partiendo de esta idea, este trabajo tiene dos objetivos principales:

1. Existen numerosas cuestiones que pueden mejorar la fase de desarrollo de software de entretenimiento, muchas de ellas centradas en **el comportamiento del personaje**, más allá del los aspectos visuales y la animación. La presente tesis pretende dar un paso adelante en los métodos y mecanismos que contribuyen a mejorar este particular desarrollo de software contribuyendo en la investigación de:

   (a) **La creación automática de controladores productores de estrategias, que sirvan como punto de partida, para los personajes**. La parte dedicada a la programación de controladores de personajes consume mucho tiempo y suele contener errores, por lo que cualquier mecanismo que pueda crear automáticamente comportamientos básicos en ciertos escenarios supondrá un ahorro de recursos en la fase de desarrollo.

   (b) Dotar de realismo al videojuego proporciona mayor capacidad de inmersión en la historia y en la trama del mismo; este aspecto va adquiriendo cada vez más importancia en la industria dedicada al desarrollo de nuevos videojuegos. La parte gráfica, que ha mostrado un avance importante, debe ser fiel a la realidad e ir acompañada de un comportamiento y conducta de los personajes creíble. Por ello, identificamos **la necesidad de dotar a los personajes de características emocionales y de personalidad** que se vean influidas por la percepción del entorno y las circunstancias circundantes.

   Para lograr este objetivo, proponemos la **creación de un modelo para introducir un mecanismo formal y estandarizado** capaz de crear automáticamente controladores generadores de estrategias y dotar de características emocionales a los personajes.

2. La arquitectura debe ser flexible, y estar abierta a la inclusión de implementaciones de controladores y modelos emocionales, para que pueda ser utilizada por diferentes enfoques compatibles con los requisitos de la arquitectura, lo que conduce a un **modelo más adaptable** que pueda beneficiarse de los conocimientos de los diferentes equipos de desarrollo, especializados en técnicas que se pueden aplicar al desarrollo a través de esta arquitectura.

3. En cuanto a la experimentación y la aplicación, es indispensable **crear un conjunto de técnicas, modelos o algoritmos**, que puedan aplicarse en la arquitectura que proponemos y que cumplan con los requisitos del entorno de aplicación, no sólo en el aspectos computacionales, sino también en el objetivo final de construir personajes más creíbles capaces de ser controlados por los comportamientos creados automáticamente. Por lo tanto, se presenta el objetivo de crear un algoritmo capaz de generar estrategias en entornos complejos (cualquier escenario de un videojuego comercial), en las cuales el estado de ánimo del personaje fija las metas. El personaje debe experimentar la emoción producida por los acontecimientos del entorno y alcanzar diferentes estados de ánimo, por ello tenemos que crear un modelo emocional, lo suficientemente flexible, que pueda ser utilizado de modo general en cualquier entorno de videojuegos y lo suficientemente potente como para representar fielmente los mecanismos emocionales.

4. Por último, es crucial **demostrar la arquitectura y los modelos en un entorno real de desarrollo de videojuegos**, teniendo en cuenta las limitaciones del proceso de desarrollo de este tipo de software: el tiempo y la complejidad. La arquitectura y los modelos que proponemos deben ser probados en productos que validen su aplicación en futuros desarrollos.

E.3 Metodología

Esta tesis aborda los objetivos descritos anteriormente: *crear una arquitectura* que incluya los componentes necesarios capaces de sostener la *creación automática de controladores* para personajes de videojuegos y dotar a estos personajes de un *comportamiento emocional* creíble.

Por lo tanto, aplicamos un enfoque descendente en el diseño de la solución.

① Creamos una descripción general de los componentes que queremos incluir (detallada en la sección 1.3.1).

② Analizamos los diferentes componentes con un enfoque ascendente extrayendo las principales características del entorno de la aplicación, porque queremos una arquitectura aplicable al campo específico del diseño y de producción de videojuegos. Esta fase detalla los procesos a realizar por cada uno de los componentes de acuerdo a un

conjunto de pre y post condiciones desarrollando una *especificación abstracta de los componentes procesales* (Secciones 1.3.2 y 1.3.3).

③ Establecemos las interfaces de intercambio de información necesarias, que otorgan la intercomunicación entre los componentes de la arquitectura. Describimos los datos de entrada requeridos por los componentes y los datos de salida por ellos producidos.

Con los tres pasos mencionados anteriormente, proponemos la arquitectura descrita en la sección 1.3.4.

④ Aplicamos la arquitectura dentro de un marco de videojuegos como ejemplo de aplicación de las técnicas para crear controladores automáticos y dotar de comportamiento emocional a los personajes en el escenario de juego.

⑤ Validamos cada uno de los componentes implementados de la arquitectura, teniendo en cuenta sus requisitos, y probamos cada uno de ellos de forma independiente.

⑥ Probamos la integración de los componentes para proporcionar una solución completa en el contexto de un videojuego.

E.4 Conclusiones

E.4.1 Mejorar la experiencia del jugador

La presente tesis se centra en la aplicación dentro de los modelos de desarrollo de software de videojuegos de los objetivos presentados en la sección 1.2. Con este fin, teniendo en cuenta otros aspectos al margen de los elementos gráficos, enfocamos la investigación en los siguientes puntos orientados a mejorar la experiencia del jugador:

1. Investigamos el campo de la psicología cognitiva con el fin de encontrar modelos aplicables dentro del desarrollo de software de videojuegos y creamos el *Modelo EEP* (descrito en el capítulo 4). Tenemos en cuenta el componente afectivo de los personajes, consideramos que supone una mejora importante incluir este aspecto en las escenas de los videojuegos para aportar mayor realismo.

2. Analizamos las técnicas de Soft Computing adecuadas a este tipo de entorno y desarrollamos el *algoritmo WEREWoLF* (detallado en el capítulo 5). La creación de los controladores de los personajes en la mayoría de los videojuegos consume mucho tiempo, tiende a contener errores de programación y puede no suponer un reto atractivo para los jugadores. Las soluciones de comportamiento estático de los personajes se puede mejorar aplicando mecanismos que las complementen. La creación automática de contenidos, en nuestro caso los controladores de patrones de comportamiento, son una tendencia a seguir por la industria del videojuego.

3. Diseñamos la *Arquitectura AGCBAR* (capítulo 3). Si bien *Modelo EEP* y el *algoritmo WEREWoLF* pueden aplicarse de manera independiente en algunos desarrollos de videojuegos, consideramos que es más fácil obtener todo su potencial si se utilizan de manera conjunta controlados por una arquitectura. La *Arquitectura AGCBAR*, por lo tanto, proporciona las características necesarias para poder aplicar el modelo EPP y el algoritmo WEREWoLF, tanto de manera aislada como conjunta, y, además, define la comunicación que debe existir entre estos y los motores de videojuegos.

E.4.2 Contribuciones al modelado emocional

El Modelado Emocional es un tema ampliamente tratado por diversas disciplinas y con variado potencial de aplicación. En esta tesis hemos centrado nuestra atención en la explotación los modelos emocionales aplicados a videojuegos. Para poder hacer encajar la modelización emocional dentro de la presente tesis, hemos seguido una serie de pasos preliminares:

1. Indentificación de los requisitos necesarios para la simulación de emociones en el campo de los videojuegos.
2. Análisis de los modelos emocionales existentes aplicados a los personajes virtuales.
3. Estudio de teorías cognitivas:
   (a) Ortony, et al. Modelo OCC para el análisis estructural de las emociones.
   (b) McRae and Costa. Cinco grandes factores de la personalidad.
   (c) Mehrabian and Russel. Modelo Temperamental PAD para la representación de estados de ánimo y emociones.

En esta tesis, contribuimos al campo de la investigación emocional desarrollando un nuevo modelo al que hemos llamado *EEP* con las siguientes características:

- define una serie de Mood Tags para etiquetar distintos estados de ánimo.
- se basa en un formalismo matemático, llamado Mood Vector Space (MVS), una extensión de un Espacio de Hilbert con las correspondientes operaciones para proporcionar la combinación de emociones y dinámica de estados de ánimo.
- incluye una estructura de análisis de emociones producida por los eventos percibidos por los personajes.
- es capaz de describir perfiles de personalidad de los personajes y traducirlos en una tendencia de ánimo de los personajes inicial y central.

Además, el *Modelos EEP* ha sido evaluadado como componente integral del motor de un videojuego comercial en dos escenarios diferentes:

1. Un escenario de combate en el cual el ánimo influye en el comportamiento de los distintos personajes. Los comportamientos están implementados como distintos controladores que manejan a los personajes dependiendo del estado de ánimo en el que se encuentren.

2. Un escenario de cómo contar aventuras en el que el argumento evoluciona a través de transiciones emocionales de una serie de peronajes no jugadores del entorno.

Por último, hemos llevado a cabo un análisis comparativo, en términos de características y capacidades expresivas, entre nuestro *Modelo EEP* y los ya existentes *EMA* y *ALMA*.

E.4.3 Contribuciones en el contexto de los controladores de aprendizaje

Como es sabido, el diseño de los controladores de personajes consume mucho tiempo y es propenso a errores, por lo que las personas dedicadas al desarrollo de videojuegos tienen que dedicar mucho esfuerzo en el diseño y la implementación del juego. Por este motivo, encontrar la manera de producir de manera automática código fiable de control de personaje es muy interesante para la industria del videojuego.

La presente tesis propone una nueva estrategia para construir controladores de personajes en escenarios de combate por medio de:

1. Una definición previa de un conjunto de controladores de referencia. Estos controladores pueden ser creados a mano o como resultado de un proceso automático previo de controladores de personajes.

2. Un proceso iterativo de entrenamiento para crear de manera automática nuevos controladores que compitan contra el controlador de referencia.

3. Extraer la nueva estrategia aprendida como un nuevo paquete de control.

El elemento clave de este proceso de entrenamiento involucra un algoritmo de aprendizaje para los controladores de personaje. En esta tesis proponemos un innovador algoritmo, *WEREWoLF*, que se basa en la combinación de dos técnicas de inteligencia computacional:

1. Computación Evolutiva. Hemos probado dos técnicas evolutivas distintas, Estimation of Distribution Algorithms (EDA) y Differential Evolution (DE).

2. Aprendizaje reforzado. Hemos considerado dos alternativas: SARSA y WoLF.

Las exhaustivas pruebas de experimentación han conducido a la creación de la nueva estrategia de aprendizaje propuesta (*WEREWoLF*) frente a los componentes de aprendizaje reforzado básicos. Los resultados obtenidos demuestran claramente que la configuración del algoritmo *WEREWoLF* usando una estrategia de Evolución Diferencial junto

con el algoritmo de aprendizaje WoLF tiene mejor rendimiento que el resto de configuraciones y algoritmos básicos. Las pruebas experimentales están respaldadas por análisis estadístico para confirmar los resultados con suficiente robustez.

E.4.4 Contribución en el contexto de desarrollo de arquitecturas de juego

Con el fin de integrar las contribuciones anteriormente mencionadas en un entorno de trabajo unificado, esta tesis propone una nueva arquitectura de componentes nueva que cubre:

1. *Dinámicas de estados de ánimo*, que engloba el componente de simulación afectiva deseado para los personajes del videojuego.

2. *Creación automática de controladores*, que tiene en consideración el proceso de creación de las diferentes estrategias de los personajes en los entornos del videojuego.

La nueva arquitectura de componentes presentada, *AGCBAR*, para proporcionar la creación automática de controladores de comportamiento con respuestas afectivas tiene dos fases de aplicación:

1. Generación Off-Line de estrategias. En esta fase radican los estados de diseño y desarrollo de la producción del videojuego y dirige la Creación Automática de Controlador.

2. Acción en tiempo de ejecución y selección de estrategias. Esta fase orquesta las Dinámicas de estados de ánimo e incluye en el personaje los distintos controladores creados automáticamente.

Por último, basándose en la *Arquitectura AGCBAR*, esta tesis proporciona un entorno completo para investigar otros muchos aspectos de la Inteligencia Artificial en videojuegos. La flexibilidad de la arquitectura le proporciona la posibilidad de sustituir cualquier componente con el fin de evaluar otras técnicas y modelos.

References

[1] C. Amato and G. Shani. High-Level Reinforcement Learning in Strategy Games. In *9th Inter. Conf. on Autonomous Agents and Multiagent Systems*, pages 75–82, 2010.

[2] G. Ball and J. Bresse. *Embodied Conversational Agents*, chapter Emotion and Personality in a Conversational Agent, pages 189–219. MIT Press, 2000.

[3] A. Bartish and C. Thevathayan. BDI Agents for Game Development. In *1st Inter. Conf. on Autonomous Agents and Multiagent Systems*, pages 668–669, 2002.

[4] C. Bartneck. How Convincing is Mr. Data's Smile: Affective Expressions of Machines. *User Modeling and User-Adapted Interaction*, 11(4):279–295, 2001.

[5] C. Bartneck. Integrating the OCC Model of Emotions in Embodied Characters. In *Proc. of the Workshop on Virtual Conversational Characters: Applications, Methods, and Research Challenges*, 2002.

[6] C. Bartneck, M. J. Lyons, and M. Saerbeck. The Relationship Between Emotion Models and Artificial Intelligence. In *SAB2008 Workshop on The Role of Emotion in Adaptive Behavior and Cognitive Robotics*, Osaka, 2008.

[7] C. Becker-Asano. *WASABI: Affect Simulation for Agents with Believable Interactivity*. PhD thesis, F. of Technology, Uni. of Bielefeld, https://www.becker-asano.de/Becker-Asano_WASABI_Thesis.pdf, 2008.

[8] E. Bethke. *Game Development and Production*. Wordware Publishing Inc., 2003.

[9] B. Boehm, B. Clark, E. Horowitz, R. Madachy, R. Shelby, and C. Westland. Cost Models for Future Software Life Cycle Processes: COCOMO^{TM}2.0. *Annals of Software Engineering*, 1:57–94, 1995.

[10] M. Bowling and M. Veloso. An Analysis of Stochastic Game Theory for Multiagent Reinforcement Learning. School of Computer Science. Carnegie Mellon University. Course Report, October 2000.

[11] M. Bowling and M. Veloso. Multiagent Learning Using a Variable Learning Rate. *Artificial Intelligence*, 136:215–250, 2002.

[12] B. Brathwaite and I. Schreiber. *Challenges for Game Designers*. Charles River Media, 2009.

[13] B. D. Bryant and R. Miikkulainen. Neuroevolution for Adaptive Teams. In *Proc. of the 2003 Congress on Evolutionary Computation (CEC 2003)*, pages 2194–2201, Piscataway, NJ, 2003. IEEE.

[14] S. Bura. High-Level Character Authoring & Utility AI in Storybricks. http://www.storybricks.com/, June 2012.

[15] B. Carolis, C. Pelachaud, I.. Poggi, and M. Steedman. APML, a Markup Language for Believable Behavior Generation. In Helmut Prendinger and Mitsuru Ishizuka, editors, *Life-Like Characters*, Cognitive Technologies, pages 65–85. Springer Berlin Heidelberg, 2004.

[16] A. Champandard. Behavior Trees for Next-Gen Game AI. In *Game Developers Conference*, 2007.

[17] H. Chandler. *The Game Production Handbook (Game Development Series)*. Charles River Media, Inc., Rockland, MA, USA, 2005.

[18] C. Claus and C. Boutilier. The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems. In *AAAI '98/IAAI '98: Proc. of the 15th Nat./10th Conf. on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, pages 746–752, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[19] A. R. Damasio. *Descartes' Error: Emotion, Reason, and the Human Brain*. Harper Perennial, 1 edition, November 1995.

[20] S. Das and P. N. Suganthan. Differential Evolution: A Survey of the State-of-the-Art. *IEEE Trans. Evolutionary Computation*, 15(1):4–31, 2011.

[21] J. Dias and A. Paiva. Feeling and Reasoning: A Computational Model for Emotional Characters. In *Progress in Artificial Intelligence*, volume 3808 of *Lecture Notes in Computer Science*, pages 127–140. Springer Berlin Heidelberg, 2005.

[22] M. S. El-Nasr, J. Yen, and T. R. Ioerger. FLAME-Fuzzy Logic Adaptive Model of Emotions. *Autonomous Agents and Multi-Agent Systems*, 3:219–257, September 2000.

[23] P. C. Ellsworth and K. R. Scherer. *Handbook of the Affective Sciences*, chapter Appraisal Processes in Emotion. Oxford University Press, 2003.

[24] E. Fossey, C. Harvey, F. Mcdermott, and L. Davidson. Understanding and Evaluating Qualitative Research. *Australian and New Zealand Journal of Psychiatry*, 36(6):717–732, 2002.

[25] D. Fradejas. Visualizador 3D y Mejoras de Funcionalidad para el Framework vBattle. Master's thesis, Universidad Rey Juan Carlos, 2013.

[26] N. H. Frijda. The Laws of Emotions. *American Phycologist*, 43:349–358, 1988.

[27] L. Galway, D. Charles, and M. Black. Machine Learning in Digital Games: a Survey. *Artificial Intelligence Review*, 29:123–161, 2008.

[28] D. Garlan. Software Architecture: a Roadmap. In *ICSE 00: Proc. of the Conf. on The Future of Software Engineering*, pages 91–101. ACM Press, 2000.

[29] P. Gebhard. ALMA: a Layered Model of Affect. In *AAMAS '05: Proc. of the 4th Inter. J. Conf. on Autonomous agents and MAS*, pages 29–36, New York, NY, USA, 2005. ACM.

[30] T. Graepel, R. Herbrich, and J. Gold. Learning to Fight. In *Proc. of the Inter. Conf. on Computer Games: Artificial Intelligence, Design and Education*, 2004.

[31] J. Gratch. Émile: Marshalling Passions in Training and Education. In *Proc. of Autonomous Agents*, 2000.

[32] J. Hu and M. P. Wellman. Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm. In *Proc. of the 15th Inter. Conf. on Machine Learning*, pages 242–250, 1998.

[33] K. Isbister and N. Schaffer. *Game Usability: Advancing the Player Experience*. Morgan Kaufmann, August 2008.

[34] J. Ito, D. Pynadath, and S. Marsella. Modeling Self–Deception within a Decision-Theoretic framework. In *8th Inter. Conf. on Intelligent Virtual Agents*, 2008.

[35] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:235–285, 1996.

[36] P. Larrañaga, R. Etxeberria, J. A. Lozano, and J. M. Peña. Optimization in Continuous Domains by Learning and Simulation of Gaussian Networks. In *Proc. of the 2000 Genetic and Evolutionary Computation Conference*, pages 201–204, Las Vegas, Nevada, USA, 2000.

[37] P. Larrañaga and J. A. Lozano, editors. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer, Boston, MA, 2002.

[38] A. LaTorre, J. M. Peña, S. Muelas, and A. Freitas. Learning Hybridization Strategies in Evolutionary Algorithms. *Intelligend Data Analysis*, 14(3), 2010.

[39] R. Lazarus. *Emotion and Adaptation*. Oxford University Press, 1991.

[40] J. Lester, J. L. Voerman, S. G. Towns, and C. B. Callaway. COSMO: A Life-Like Animated Pedagogical Agent with Deictic Believability. In *Proc. of the IJCAI97 Workshop on Animated Interface Agents: Making them Intelligent*, 1997.

[41] S. M. Lucas and G. Kendall. Evolutionary Computation and Games. *IEEE Computational Intelligence Magazine*, 1:10–18, February 2006.

[42] C. Madeira, V. Corruble, G. Ramalho, and B. Ratitch. Bootstrapping the Learning Process for the Semi-Automated Design of a Challenging Game AI. In *AAAI 2004 workshop on Challenges in Game AI*, pages 72–76, 2004.

[43] D. Maravall, J. de Lope, and J. A. Martin. Hybridizing Evolutionary Computation and Reinforcement Learning for the Design of Almost Universal Controllers for Autonomous Robots. *Neurocomputing*, 72(4–6):887–894, 2009.

[44] S. Marsella, J. Gratch, and P. Petta. *A Blueprint for Affective Computing: A Sourcebook and Manual*, chapter Computational Models of Emotions, pages 21–47. Oxford University Press, 2010.

[45] S. Marsella, W. L. Johnson, and C. LaBore. Interactive Pedagogical Drama. In *Int. Conf. on Autonomous Agents, Agents*, pages 301–308, 2000.

[46] S. C. Marsella and J. Gratch. EMA: A Process Model of Appraisal Dynamics. *Journal of Cognitive Systems Research*, 10:70–90, 2009.

[47] C. Martinho. Emotions in Motion: Short-Time Development of Believable Pathematic Agents in Intelligent Virtual Environments. Master's thesis, Universidade Tecnica de Lisboa, 1999.

[48] R. R. McCrae and P.T. Costa. *The Five-Factor Model of Personality: Theoretical Perspectives*, chapter Toward a New Generation of Personality Theories: Theoretical Contexts for the Five-Factor Model, pages 51–87. 1996.

[49] A. Mehrabian. Framework for a Comprehensive Description and Measurement of Emotional States. *Genetic, Social, and General Psychology Monographs*, 121:339–361, 1995.

[50] A. Mehrabian. Analysis of the BigFive Personality Factors in Terms of the PAD Temperament Model. *Australian Journal of Psychology*, 48(2):86–92, 1996.

[51] A. Mehrabian. Pleasure-Arousal-Dominance: A General Framework for Describing and Measuring Individual Differences in Temperament. In *Current Psychology*, volume 14, pages 261–292, 1996.

[52] A. Mehrabian and J. A. Russell. *An Approach to Environmental Psychology.* The MIT Press, 1974.

[53] H. Mühlenbein and G. Paaß. From Recombination of Genes to the Estimation of Distributions I. Binary Parameters. *4th Inter. Conf. on Parallel Problem Solving from Nature*, 1141:178–187, 1996.

[54] A. Moors, J. De Houwer, and P. Eelen. Unintentional Processing of Motivational Valence. *Quaterly Journal of Experimental Psychology. A, Human Experimental Psychology*, 58(6):1043–1063, 2005.

[55] L. B. Morelli and E. Y. Nakagawa. A Panorama of Software Architectures in Game Development. In *23rd Inter. Conf. on Software Engineering & Knowledge Engineering*, pages 752–757, 2011.

[56] D. Moriarty, A. Schultz, and J. Grefenstette. Evolutionary Algorithms for Reinforcement Learning. *Journal of Artificial Intelligence Research*, 11:241–276, 1999.

[57] W. N. Morris. *Mood: the Frame of Mind*. New York: Springer-Verlag, 1989.

[58] A. Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.

[59] W. T. Norman and L. R. Goldberg. Raters, Ratees and Randomness in Personality Structure. *Journal of Personality and Social Psychology*, 4:681–691, 1966.

[60] A. Ortony, G. L. Clore, and A. Collins. *The Cognitive Structure of Emotions*. Cambridge University Press, 1988.

[61] R. J. Palma-Duran. *Java Behaviour Trees, User Guide*. Universidad Complutense de Madrid, http://sourceforge.net/projects/jbt/, September 2010.

[62] L. Peña, A. LaTorre, J. M. Peña, and S. Ossowski. Tentative Exploration on Reinforcement Learning Algorithms for Stochastic Rewards. In *4th Inter. Conf. Hybrid Artificial Intelligent Systems*, pages 336–343, 2009.

[63] L. Peña, S. Ossowski, and J. M. Peña. vBattle: A New Framework to Simulate Medium-Scale Battles in Individual-per-Individual Basis. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 61–68, Sept. 2009.

[64] L. Peña and J. M Peña. Mood Vectorial Space: Formalism. Technical report, Univertity Rey Juan Carlos, 2010.

[65] L. Peña, J. M. Peña, and S. Ossowski. Representing Emotion and Mood States for Virtual Agents. In *German Conference on Multi-Agent System Technologies (MATES)*, 2011.

[66] L. Peña, J. M. Peña, S. Ossowski, and P. Herrero. Evolving Q-Learners for Stochastic Games: Study on Video Game Agent Controllers. In *World Automation Congress (WAC)*, 2010.

[67] L. Peña, J. M. Peña, S. Ossowski, and S. M. Lucas. Learning and Evolving Combat Game Controllers. In *2012 IEEE Conference on Computational Intelligence and Games*, number 2012, Granada Spain, 2012.

[68] L. Peña, J. M. Peña, S. Ossowski, and J. A. Sanchez. EEP – a Lightweight Emotional Model: Application to RPG Video Game Characters. In *2011 IEEE Conference on Computational Intelligence and Games*, 2011.

[69] David Pereira, Eugénio Oliveira, and Nelma Moreira. Formal Modelling of Emotions in BDI Agents. In Fariba Sadri and Ken Satoh, editors, *Computational Logic in Multi-Agent Systems*, volume 5056 of *Lecture Notes in Computer Science*, pages 62–81. Springer Berlin Heidelberg, 2008.

[70] F. Petrillo, M. Pimenta, F. Trindade, and C. Dietrich. What Went Wrong? A Survey of Problems in Game Development. *Computers in Entertainment*, 7(1):13:1–13:22, February 2009.

[71] M. Pfeiffer. Machine Learning Applications in Computer Games. Master's thesis, Institute for Theoretical Computer Science, Graz University of Technology, http://www.igi.tugraz.at/cluster/pfeifferDA.pdf, 2003.

[72] R. Picard. *Affective Computing*. MIT Press, 1997.

[73] I. J. Roseman, A. A. Antoniou, and P. E. Jose. Appraisal Determinants of Emotions: Constructing a More Accurate and Comprehensive Theory. In *Cognition and Emotion*, volume 10, pages 241–277, 1996.

[74] J.A. Russell and A. Mehrabian. Evidence for a Three-Factor Theory of Emotions. *Journal of Research in Personality*, 11:273–294, 1977.

[75] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.

[76] D. Sander, D. Grandjean, and K. R. Scherer. A Systems Approach to Appraisal Mechanisms in Emotion. *Neural Networks*, 18:317–352, 2005.

[77] K. R. Scherer, T. Bnziger, and E. Roesch, editors. *A Blueprint for Affective Computing: A Sourcebook and Manual*. Oxford University Press, 2010.

[78] K. R. Scherer and P. Ekman. *Approaches to Emotion*, chapter On the Nature and Function of Emotion: a Component Process Approach., pages 293–318. Hillsdale, 1984.

[79] L. S. Shapley. Stochastic Games. *Proc. of the National Academy of Sciences*, 39(10):1095–1100, 1953.

[80] C. A. Smith and L. Kirby. *Feeling and Thinking: The Role of Affect in Social Cognition*, chapter Consequences Require Antecedents: Toward a Process Model of Emotion Elicitation. Cambridge University Press, 2000.

[81] C. A. Smith and R. Lazarus. *Handbook of Personality: Theory and Research*, chapter Emotion and Adaptation, pages 609–637. L. Pervin, 1990.

[82] K. O. Stanley. *Efficient Evolution of Neural Networks Through Complexification*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, 2004.

[83] K. O. Stanley and R. Miikkulainen. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[84] R. Storn and K. Price. Differential Evolution - a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces. ftp://ftp.icsi.berkeley.edu/pub/techreports/1995/tr-95-012.pdf, 1995.

[85] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*, volume 9. MIT Press, http://webdocs.cs.ualberta.ca/ sutton/book/ebook/, 1998.

[86] J. Togelius, R. De Nardi, and S. M. Lucas. Towards Automatic Personalised Content Creation for Racing Games. In *Proc. of IEEE Symposium on Computational Intelligence and Games*, pages 252–259, 2007.

[87] R. J. Urbanowicz and J. H. Moore. Learning Classifier Systems: a Complete Introduction, Review, and Roadmap. *Journal of Artificial Evolution and Applications*, 2009:1:1–1:25, January 2009.

[88] G. van Lankveld, S. Schreurs, P. Spronck, and J. van den Herik. Extraversion in Games. In *Computers and Games*, volume 6515 of *Lecture Notes in Computer Science*, pages 263–275. Springer Berlin - Heidelberg, 2011.

[89] C. J. C. H. Watkins and P. Dayan. Q-Learning. *Machine Learning*, 8(3):272–292, 1992.

[90] S. Whiteson and P. Stone. Evolutionary Function Approximation for Reinforcement Learning. *Journal of Machine Learning Research*, 7:877–917, 2006.

[91] G. N. Yannakakis. *AI in Computer Games: Generating Interesting Interactive Opponents by the use of Evolutionary Computation*. PhD thesis, University of Edinburgh, 2005.

[92] M. Yoshikawa, T. Kihira, and H. Terai. Q-Learning Based on Hierarchical Evolutionary Mechanism. *WSEAS Transactions on Systems and Control*, 3(3):219–228, 2008.