MASTER THESIS

Game World Implementation of Artificial Recognition System Model

Submitted at the
Faculty of Electrical Engineering and Information Technology,
Vienna University of Technology
in partial fulfilment of the requirements for the degree of
Electrical Engineering

under supervision of

Prof. Dr. Dietmar Dietrich
Institute number: 384
Institute of Computer Technology
Vienna University of Technology

and

Dipl. Ing. Alexander Wendt
Institute number: 384
Institute of Computer Technology
Vienna University of Technology

by

Ernest Ortuño Torra
Matr.Nr. 1129171
Rötzergasse 73/23/C
A-1170 Vienna

August 2012 _____

Abstract

Artificial Intelligence (AI) has been improving and developing the last thirty years with different classifications of patterns. Although these kinds of methods have been studied and improved, they are not following the proper meaning of what should be AI because of probabilities. Sometimes, an option that has a low probability to be done could be the proper one to be chosen. However, if probabilities want to be set aside, new prioritization systems would be found as could be the psychoanalytical concept of drives. For this reason, a humanlike behaviour is required in order to achieve this kind of reasoning. The work presented in this thesis shows how it is possible to integrate all of these mental processes in an embodied agent.

Working in this way, ARS (Artificial Recognition System) research group have developed a model that combines psychoanalytical and neurology theories with methods of AI. Until now, only primitive simulations were used without any relation to real world applications. In the course of this work, an interface between a complex world, that is Unreal Tournament 2004, and ARS engine is designed and implemented to investigate if it is possible to use ARS decision unit to an external world. In particular three aspects are involved in this; firstly, the design of the interface that will provides a connection between ARS model and UT2004. Secondly, the way of how the agent will be able to detect walls that will be a part of the environment interface. Thirdly, the creation of an output interface that will be responsible to get from ARS engine the selected actions and process them. After this work, a future extension of it could be the implementation of more accurate methods that would permit UT2004 bot has a closer humanlike behaviour. Moreover, this work will deliver general interfaces that shall be used also in other applications.

Preface

When Dr. Dietmar Bruckner purposed the option of taking part in a project related to Artificial Intelligence I felt that I could not reject this opportunity for different reasons. The first reason was the challenge of create an agent that was able to take decisions in different situations by herself. The second reason was opportunity to study Sigmund Freud works, more specifically, his theories about how human mental works and how he splits human brain. Finally, I appreciated the idea of taking part of a research group who have been developing an Artificial Intelligence model since 1999 and being able to learn as much as possible of them.

Acknowledgements

To everybody who shares university period with me, especially to them who make me smile. And also to Enrique Muñoz, project mate in ARS research group.

I want to distinguish support of two members of the research group: The first one is Alexander Wendt, who provided me all information and all explanations that was required for developing this master thesis. The second one is Clemens Muchitsh, who was always ready to give his support in order to solve different technics problems related to informatics stuff. I am totally sure that without their help this master thesis could not be possible to be done. I do not want to leave this acknowledgement section without giving my gratitude to Universitat Politècnica de Catalunya, in concrete, to Escola Tècnica Superior de Telecomunicacions de Barcelona to bring me the opportunity to realize this master thesis in Technische Universität Wien.

Table of contents

Abbreviations

ARS	Artificial Recognition System
AI	Artificial Intelligence
ANN	Artificial Neural Networks
BDI	Body Desired Intention
BT	Behaviour Tree
DMS	Decision Making Systems
ICT	Institute of Computer Technology
IDE	Integrated Developer Environment
UT2004	Unreal Tournament 2004

1. Introduction

AI (Artificial Intelligence) can be defined as the ability to think for a no alife agent (such as machine). John McCarthy coined this term in 1955 identifying it as "the science and engineering of making intelligent machines". Obviously, during last half century this term has been deepened and amplified by a huge quantity of researcher groups who have worked on it giving to the society several of applications.

Although these projects, that sometimes became commercial applications, can be found in almost all knowledge fields, basically became a basic tool for the technology industry, that try to give solutions for computer science problems. Simple example is an automated online assistant who provides customer service on a web page that was one of many very primitive applications of artificial intelligence.

Since John McCarthy had started to work on it, a huge group of tools have been used and developed in order to improve AI concept. Most of them are probabilistic methods due to AI problems are directly related to uncertain information. An example is Bayesian Networks that are probabilistic graphical models that represent a set of random variables and their dependencies. They are used to analyze process in real time.

Other tool used in AI field is Artificial Neural Networks (ANN). This concept consists of an interconnected group of defined artificial neurons that processes information using what is known as a connectionist approach. At this point, the work done by a group of researchers, among them Paul Werbos, related to back propagation algorithm. It has become a common training artificial neural networks' method in order to minimize the objective function. ANNs are used to model relationships between inputs and outputs as well as to find patterns in data.

Going further in AI concept, some criticisms were made based on the impossibility for a not alive agent to act completely like a human being. However, as Howard Gardner defines in 1983, there are multiple intelligences rather than a single ability, in other words; he differentiates into various kinds of intelligence such as musical or logical-mathematical. So, first of all, it is a requirement to delimit what kind problems will be solved and how to do it. Nevertheless, this work tries to overthrow giving to a cognitive agent a humanlike behavior.

1.1 Motivation

For ten years ago, the ARS (Artificial Recognition System) research group has been working on an AI model at ICT (Institute of Computer Technology) in Vienna. This model goes in a different direction compared to previous AI projects. First of all, we have to differentiate between two AI focuses [Ole95]. The first one is to use the power of computers to augment human thinking. It means that the goal of AI is only give support and benefit to human nature without taking care of how it is done. The other one is to use a computer artificial intelligence to understand how humans think.

Traditionally, projects related to the AI field were developed behind the first focus such as robotics and expert systems [Mill88]. It was the common idea of what should be AI. Making a step forward, the desired result of ARS research project is a technically feasible psychoanalytically inspired model and not the individual experience of an artificial agent. In other words, ARS is designed to use sensitive perceptions (as inputs) to give corrects reactions (as outputs) of an embodied cognitive agent.

Once the motivation of this work has been explained, some questions could rise such as why has the need of combine these two environments (ARS and UT2004) been thought? The answer is to be able to get the possibility to test psychoanalytical usecases in a humanlike world. Meanwhile this testing is done, the bases of the first ARS model application have been built in order to make it possible one day. Among other results, it can show how well the ARS agent is suited for a computer and educational game.

1.2 Problem Description

As it seems logic, in this kind of projects is totally necessary to have an embodied agent [Wied09]. When we talk about perceptions and, as in this case are explained; sensitive or sensorial perceptions, the necessity of having a body to capture this information is evident otherwise the implementation of this AI model would not be possible. For instance, if a bot (a player, that is controlled by computer) is been shot, it need to have a body in order to know how heath level is loosed according what part of its body is shot, from where has been shot,… and, as consequence, it will be able to do as much as it can to get over whatever situation.

There are more requirements apart from disposing of an embodied agent. Necessity of having a multi-field research group is mandatory due to the requirement of arranging a good approach of human mind. Related to this, in the ARS project there are, apart from engineers, two experts in psychoanalysis, who are responsible to make the first approximation of human mental process in order to be able to implement it as much as it is possible into a code. The figure 1.1 shows what the way of working of ARS research group is.

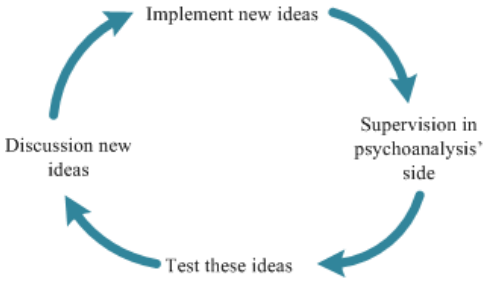


Figure 1.1: Working diagram

Until now, ARS project was implemented using that psychoanalyst model and tested through a 2D world where ARS agent could find a static object. These objects can be associated with three entities: food, obstacles and a toilet. Each of the entities satisfice each embodied agent necessities. The challenge of this master thesis is engaged this psychoanalyst model into a more complex environment.

1.3 Task Description

Now, the challenge is to test ARS model in 3D complex environment. This environment is an open source game that permit interact in an easy way with it due to it is possible introduce an external code. This open source game is called UT2004 (Unreal Tournament 2004). In this master thesis it will be designed a UT2004 bot, in other words, a code will be implemented in order to be able to watch an ARS agent interacting with the game environment.

It is expected that an embodied agent will be able to act as a human in this world and to take its own decisions based on the drives have been created as internal needs to overcome challenges. It was always the ARS research group goal. Basically two general tasks must be work:

- An interface must be created in order to connect both platforms: Unreal Tournament 2004 and ARS decision unit.
- An adaption has to be done in the ARS engine due to get a good communication, for instance, some actions (fire, jump) has to be created as well as UT2004 objects (health-pack, weapons).

All the steps that work on AI field is useful due to it can be the key to achieve a new generation of embodied agents. Of course, in this master thesis the project will permit to integrate the ARS project into a computer game and it is not the most useful field on human evolution but it can be a start point to implement this kind of model in other real environments in the real world. It is the first implementation of ARS in a commercial application. Until now, only primitive simulations were used without any relation to real world applications. As said before, this project is a step forward and it should be the direction of the projects that will come on the future and for that reason, it is a challenge to be able to participate on it and make as much as possible to improve it.

The purpose of this work is to investigate if it is possible to use ARS decision unit to an external world and how to do it. Firstly, to do that, one interface should be designed and implemented. An entity should translate inputs from Unreal Tournament 2004 to ARS engine and it should make the same operation but in the other way with outputs. It is important to define the interfaces in a manner that can be used as templates for future applications of ARS. Secondly, an UT2004 bot should be implemented to get the proper behavior required.

The exact contribution to the global ARS project of this master thesis involves three main tasks. Firstly, the design of the interface that will provides a connection between ARS model and UT2004. Secondly, the way of how the agent will be able to detect walls that will be a part of the environment interface. Thirdly, the creation of an output interface that will be responsible to get from ARS engine the selected actions and process them. The other part of the work is done in other explained in a project mate Enrique Muñoz [Mun12].

The result should be to achieve a humanlike behavior of the UT bot. It will be achieved with use-cases, that will verify how the embodied agent interacts with the complex world. For instance, a use-case will be to pick up a health pack when it will be hurt. Moreover, this work will deliver general interfaces that shall be used also in other applications.

Inside ARS model, two tasks have to be done; the first one is to create Unreal Tournament actions that are needed in order to get a proper performance. The second one is declare all the objects that UT bot can find in the complex world. Once it will be done, this interface will give the possibility to communicate ARS engine with different environments. It exist the possibility to participate in a competition called Botprize, that challenges programmers/researchers/hobbyists to create a bot for UT2004 (a first-person shooter) that can fool opponents into thinking it is another human player.

Finally, it is important to remark that among the goals of this master thesis, there is not the task to program anything in ARS model. It mean that ARS model was as a black box for this project due to it is not a requirement to improve it directly.

1.4 Methodology

When the motivation, problem and task description were clarified, it was time to start thinking about how this whole could be done. Obviously, the process of analysing platforms has required a huge amount of time. Among these platforms, there are three that must be pointed out: UT2004, Pogamut and ARS. All of them were analysed and described or used in this document among all the sections.

Once this first step was complete, the next step was imposed by the workflow: requirements generation for the ARS model. In other words, an adaptation of the ARS IO system was adapted in order to optimize the communication between the implemented model with an external world, in this case the UT2004. On the other hand, the information that was provided by the UT2004 was adapted to get a proper functionality of them.

Finally, the last step was the implementation of the interface that was responsible to connect these two environments. Using the Eclipse IDE to develop it and following the requirements arise for it,

that were thought during the process of analysing platforms. When it was implemented, the usecases were designed in order to test the proper functionality of the whole environment.

1.5 Limitations

Some difficulties will appear in the course of this work; for example; the simple fact of some actions and objects should be created inside the ARS engine in order to get a stable communication between both sides or ARS inputs are so concrete, for this reason it has to be designed well how UT dates are captured and processed and what is the best way to do it.

Apart from these problems, during the course of this work some limitations have been set in order to don't lose the focus on the main task of this master thesis and also due to some features that are out of control such as time or the point that direct modification of ARS model is not permitted so every time that some change have been done implied a stop of our work until it was done.

Due to all of these difficulties and limitations, this work only set the pillars to export ARS model to other complex environments, in this case is UT2004 but it can be other computer games such as The Sims. In other words, the result of this master thesis is not to create an Unreal Tournament bot that could be confuse with a human player using ARS as a decision making but it provides the required tools to achieve it. Also a group of simple actions are set in order to let Unreal Tournament bot be able to survives in a war environment.

2. State of the art and Related Work

When a huge work is done, a huge research must be done as well. Because of that, the claim of this chapter is to show what was used during the course of this work in order to develop it. Two main sections have been developed: *State of the art* and *related work.*

2.1 State of the art

State of the art is a concept which involves all the stuff done before Artificial Recognition System model started to be real. There are two main blocks: *Body Desired Intention* and *SOAR*. Each of these blocks is explained in a subsection.

2.1.1 BDI: Body Desired Intention

A basic concept that will be used in the ARS work and, as a consequence, during the course of this work is a particular type of rational agent. As is known, in different fields such as control of air traffic systems or telecommunications networks require a system that is able to realize control tasks and to take decisions in dynamic environments. The goal is to achieve an architecture that treats the system as a rational agent. It means that the agent has mental attitudes of Belief, Desire and Intentions (BDI) representing the information, motivational and deliberative states of the agent.

BDI Model

Some limitations and requirements are established when a BDI agent is wanted to be introduced into a system and its environment. Firstly, there are potentially many different ways in that the environment can evolve. This first idea implies that the environment is nondeterministic, so it can change over the time in unpredictable ways. Secondly, there are potentially many different actions the system can execute. In this case, the system is nondeterministic as well. Thirdly, there are potentially many different objectives that the system is asked to accomplish. For instance, a desire of nourish and a desire of attack can be asked to be fulfilled but not all of that may be simultaneously achievable. Fourthly, the actions achieve the various objectives are dependent on the state of the environment and are independent of the internal state of the system. Then, the environment can only be sensed locally so one sensing action is not sufficient for fully determining the state of the entire en-

vironment. Finally, the rate at that computations and action can be carried out is within reasonable bounds to the rate at that the environment evolves.

Three concepts have a main importance inside the BDI model [Huh98]. First, an input data required by the system is information related to the state of the environment. It is necessary that there be some component of the system that can represent this information and update it after each sensing action. This first concept is known as system beliefs. Second, it is also necessary that the system have information about the objectives to be accomplished. This component is called system desires. Finally, a component of system state to represent the currently chosen course of action is needed. This last concept is known as system intentions. Basically, the intentions of the system have a huge impact on the deliberative component of the system.

One of the first ideas to realize an implementation for modelling the behaviour of such a system was a branching tree structure. The principle was easy: each branch in the tree represents an alternative execution path, where each node in the structure represents certain state of the world. Moreover, a differentiation must be done between the action taken by the system and the events taking place in the environment, so two concepts must be defined: choice nodes, representing the options available to the system, and chance nodes, representing the uncertainty of the environment.

While it is not necessary [Ein03], assuming that the events and actions are atomic and deterministic during loop of the system is a useful practice. If not a faster model must be created in order to reduce the risk of a significant event occurring during computation. Basically, a BDI model must follow the idea of the pseudo code that is shown in the next figure.

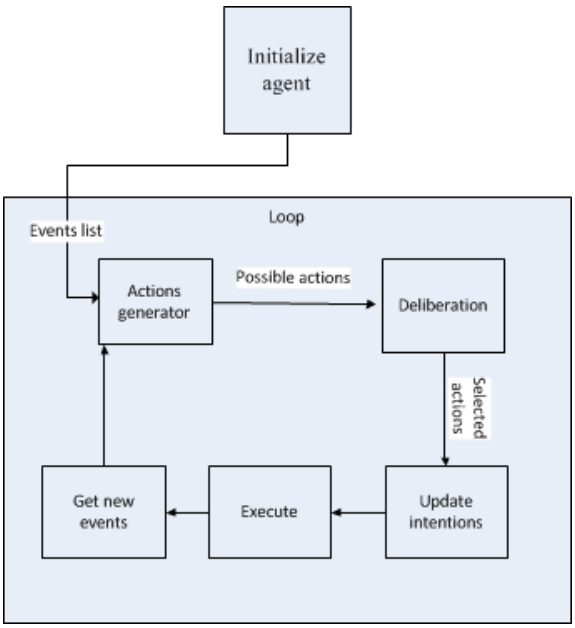


Figure 2.1: BDI workflow [Ein03]

To sum up, the architecture BDI let the machines think as human beings, imitating the human decision making processes and, as a result, make the agent play in a human like way. Obviously, it gives the possibility to create a computer games characters, also known as bots, appear as realistic as possible and, as a consequence, getting a better experience of the interaction between a human player with a computer player.

Application

In the previous section, an idea of what is the theory behind BDI is developed and now it is time to show the ideas of some BDI implementations in different systems and environments that will be exposed in this section. In particular, one application is show to gives a general idea of the BDI concept.

OASIS

One of them was an air-traffic management system, OASIS [Rao95]. The whole system was working receiving live information from the Sydney airport radar. It was inspirited from philosophical theories of Bratman who argues that intentions play a significant and distinct role in practical reasoning and cannot be reduced to beliefs and desires.

The main objective of OASIS was to realize an optimal sequence to land all aircraft in a save conditions. One of deals that were treated in this project was the uncertain in the data related to the wind. The system was generating a sequence of waypoints that were send to each aircraft. An aircraft was identified as an agent in the environment.

This project based its succeed on two main pillars: the ability to react and the balance in the behaviour agent between reactive and goal-directed. Although ability of reaction implies a good environment acknowledge, the agent build plans starting from his purpose and, of course, it is able to be sensitive to the current context. The point to get a balanced behaviour was thanks of the periodically reconsideration of committed plans.

Working in BDI

One of all researchers who are working in the field of AI in embodied agents and, more specific, in the BDI model developing area is Frank Dignum who is playing an important role since some years ago from Universitat of Utrecht. Since September 2000, he is an associate professor at the Decision Support group of the Insitute of Information and Computing Science. Also he made some collaboration works at the Department of Information Systems of the University of Melbourne.

Frank Dignum focuses his interest [Dig09] on formal specification of social aspects and reliable and flexible communications between autonomous systems. Another field where he is spending some time is in electronic commerce, where agent based social simulation and serious gaming have a huge impact. He thinks that the best way to understand how systems are implemented is to have a formal specification of systems. Because of that, he uses some form of dynamic logic that incorporates both deontic elements as well as communication actions in her research.

Although the use of BDI agents for behaviour modelling of virtual humans in games, in some application use of communication by agents or each itself is essential and can be the main driving force for their behaviour and therefore the advancement of a story or scenario in terms of mental and environmental changes. Usually the focus is on non-communicative agent behaviour like navigation or object manipulation while interactions with the player or trainee are limited.

CARIM

Figure 2.2: CARIM screenshot [Dig11,pag 41]

CARIM project was one of the lots of projects that play with all concepts related to BDI agents. The main porpoise of this project was the creation of intelligent agents that would be used for team-based command and control simulation training [Dig11]. The BDI agent paradigm provides a complete framework used by CARIM researchers group due to facilities an efficient workflow converting cognitive models to corresponding BDI concepts like beliefs, desires and intentions. A screenshot of CARIM is shown in the next picture.

In the next figure a graphic idea of CARIM logic. In the project, two main domains were specified: game engine, multi-agent system. The agent physical layer executes the actions that have been scheduled by the cognitive layer, influencing the environment. Changes in the environment are sensed in the physical layer and communicated back to the cognitive layer. As will be explained later, this figure could be representative for the work exposed in this master thesis where the cognitive layer is ARS decision unit, environment is Unreal tournament 2004 and physical layer is the interface designed and implemented in this work.

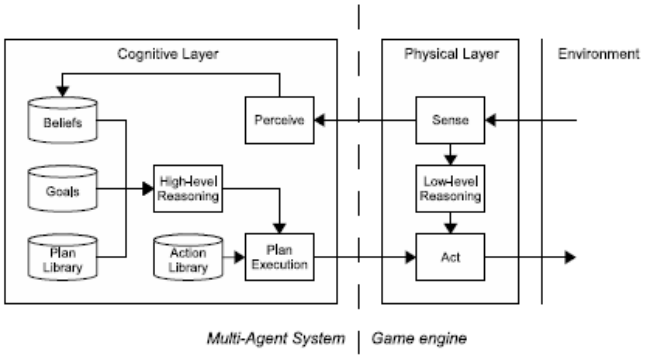


Figure 2.3: CARIM environment model [Dig11, pag 42]

The dialogue system has provided the communicative behaviour required for the CARIM fire-fighting training system, showing that the BDI approach to model communication for agents in training simulations is both useful and fruitful. By creating agents that can communicate naturally we reduce the dependence on many human key role players in training scenarios. Although communication can now be employed by agents to achieve a desired result, it is not always possible, desirable or the most efficient way to achieve some goal.

2.1.2 SOAR

Soar is a project [Lai12], that was developed by John Laird, Paul Rosenbloom, and Allen Newell 1986 in Michigan University, providing a general cognitive architecture for developing systems. These systems require be able to show an intelligent behaviour and to work using cognitive science. One of these systems is Artificial Recognition System (ARS) that uses SOAR to support all the capabilities required of a general intelligent agent when it tries to develop in a specific environment.

Basically, Soar is designed on the basis of the hypothesis that all deliberate goal –oriented behavior can be cast as the selection and application of operators to a state. A state is a representation of the current problem-solving situation; an operator transforms a state (makes changes to the representation); and a goal is a desired outcome of the problem-solving activity. In fact, Soar is continually trying to select and apply operators as figure 2.4 shows.

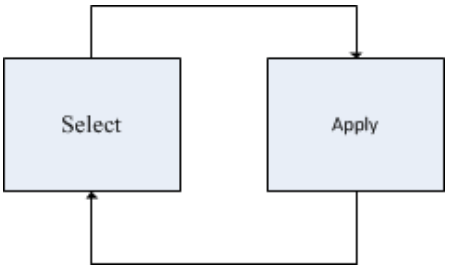


Figure 2.4: Soar execution[Lai12, pag 5]

In the previous figure, Soar execution is shown how two main concepts are repeated: select and apply. In order to select an operator, three kind of knowledge must be known. The first is operator proposal that represents if an operator is appropriate for the current situation. The second is operator comparison that is knowledge to compare candidate operators. The third concept is operator selection. It is related to knowledge to select a single operator based on the comparisons. When an operator wants to be applied, knowledge of how a specific operator modifies the state must be known, that is called operator application. Finally, another type of knowledge in shared by the selection and the apply process: knowledge of monotonic inferences that can be made about the state.

Fundamental questions were raised during the course of the project. For instance one of them was whether learning process will go on forever and how much of learned knowledge is used. To understand these issues better, Soar was instrumented to let to researchers collect data and characterize its long-term behaviour. Then, based on an understanding of these characteristics, Soar was modified to address the utility problem.

2.2 Related Work

One thing is the previous work and another completely different is the related work. In other words, all the stuff used to set pillars of this master thesis. There are four main blocks: *Psychoanalysis*, *Artificial Recognition System model*, *Unreal Tournament 2004* and *Pogamut*. Each of these blocks is explained in a subsection.

2.2.1 Artificial Recognition System Model

The second pillar of this project is the Artificial Recognition System model due to it is the model used to give to a cognitive agent the possibility to think and act as a human being. In this section a theoretical and technical approach will be done in order to go deeper inside the model.

Psychoanalysis

One of the main goals is to achieve a cognitive agent that will be able to have a humanlike behaviour. To do that, a look into psychoanalysis field must be taken in order to understand how human mind works and how a model can emulate it.

SIGMUND FREUD

If one name is chosen to sum up the concept of psychoanalysis or, what is the same, what we today known as the science of psychology, it would be *Sigmund Freud* (1856-1939). Although at first he was trained as a physician, soon he started to get deeper in other field related to the mind, as the fact shows that he began research career studying animal biology. Later, he started to become interested in hypnosis, thanks to Joseph Breuer, and, as a consequence of it, about something that Breuer called *talking cure*, that is an extended practice nowadays. In Freud work, it exist a clear emphasis on the sexuality role [Ahm12].

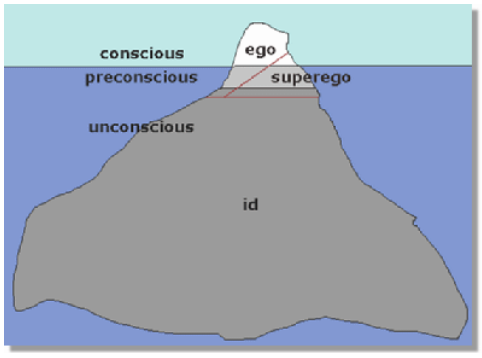
The practice of hypnosis to treat her patients was used by Freud in order to bring out repressed traumatic memories of them. He was convinced that unconscious forces, that the individual is unaware, were jamming the psychical patient conscious system. The point, according to Freud, is that most memories are not available in consciousness and if a painful memory is wanted to be repressed generates a continuously tension feeling.

Freud realized that no everybody was able to be hypnotized. Because of that, he developed another technique: free association [Rid06]. The point was that patient said the first word that appears in his mind. In some of these words generates an association with unconscious conflict. These associations are interpreted by analyst who reflects them back to patient.

Moreover, Freud followed the idea of dreams represented wishes that are not satisfied by patients. Most of them are not allowed to be satisfied by social rules or conscious mind. Two parts can be identified in a dream: manifest dream and latent dream. Manifest dream, that is described by patient, is dream part that can be remembered in consciousness. On the other hand, latent dream, that is treated by analyst, represents true meaning of the dream.

A concept, that will have a huge impact in ARS model, was defined by Freud: drive. A drive represents an instinct or impulse that attempt to avoid pure reductionism to physiological forces. Two main groups of drive can be identified: Eros (life impulses) and Thanatos (death impulses). Life impulses are associated to survival instincts. Freud created a term for psychic energy derived from sexuality called libido. On the other hand, death impulses are the source of aggressiveness.

One of the main works in Freud career and, as a consequence in psychoanalysis theories, was the psychoanalytic structure of personality [Boe97]. Freud establishes a terminology to explain how personality works: id, ego and superego. As the next figure shows, these three concepts work between the consciousness and the unconsciousness.



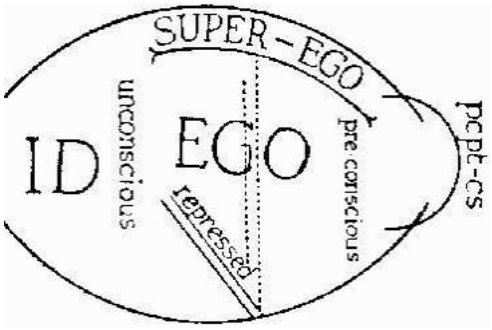


Figure 2.5: Freud structure [Fre33, p. 77] **Figure 2.6**: Freud structure [Fre33, p. 77]

The first entity is known as Id. It generates primary necessities of human body, so it is the first personality structure that develops in human beings and it remains in the unconsciousness. It can be said that is the most primitive, disorganized and innate of the personality. The Id proposes is re-

duced, as much as is possible, the tension created for primitive pulse. For instance are related to hungry, sex, impulsive reactions. It acts according to the pleasure principle and know nothing about reality demands.

The second entity is called Super-Ego. It can be seen like the opposite of the Id. It involves everything related to moral conscience and social rules. Freud theory implies that the super-ego is a symbolic internalization of the father figure and cultural regulations. It acts as the conscience maintaining our sense of morality and proscription from taboos. Super-Ego consists in two main components. The first component is conscience or capacity for self-evaluation, criticism and reproach. The second one is an idealized self-image in the eyes of societal rules.

The last entity is Ego. It has the responsibility to achieve in a real way wishes and demands from Id to external world, and in the same time, following demands from Super-Ego. So, it can be seen as a "referee" between the other two entities. It is based on cognitive and perceptual skills that distinguish fact from fantasy, allowing the ego to satisfy id needs in an appropriate manner.

Sometimes, the ego is not able to satisfy demands of the id or superego. When it happens the ego has what is known as defence mechanisms, that are a short-term solution. For instance one of these defence mechanisms is repression. It is characterized by blocking a wish or desire from the consciousness. This situation can happen when a human is unaware of deep-seated anger. Another example is rationalization that is when a human deals with an emotion to avoid the upset. For instance when a human thinks as an excuse: "everybody does it, why feel bad".

Some critics appear against the classical psychoanalysis later [Rob93]. For instance, Freud said that neuroses were the result of childhood sexual abuse but later this theory was rejected owing to criticism from contemporary society. Moreover, a focus on development on male gender can be found in Freud work. It implied that female development either mirrored male development or was inferior. Behind this idea, Freud used to say: "Anatomy is destiny" or "Woman is as deficient man (lacking penis)". Of course biology does not support the idea of female as castrated male.

PSYCHOANALYSIS IN ARS MODEL

As have been said in introduction, ARS model have been designed on the basis of psychoanalysis. This technical psychoanalytical model, that permits to achieve a humanlike behaviour in an embodied agent have been developed with the collaboration of psychoanalysts. What is more, their presence in the research team was essential in order to set the basis to start developing and implement ARS model in a complete approach.

The main reason was that using this model it would be apparently the best one in order to implement a top-down model. As it will be explained before, Freud developed a therapeutic technique to help people who suffered mental disorders and created a theory of the mind and the conduct of humans. Freud tried, in a good way, to give to the concept of unconscious a status of scientific.

Following Freud second topographical model, that describes human mental behaviour, is possible implement a top-down model [1]. As all top-down models, the first layer must be as abstract as pos-

sible. In successive layers would implemented different functionalities that will implement all need in order to achieve an accurate and reliable psychoanalytical model.

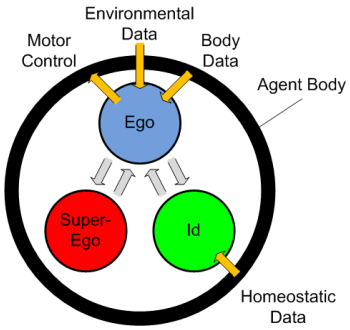


Figure 2.7: Id, Ego and Super-Ego [1]

In the step of converting the theoretical model in a model to be realistically implemented, the first layer of the ARS model was designed as the figure shows. In the second topographical model, Freud split human mind in three entities, that each one represents a mode of operation of the mind: id, ego and superego. Each of these entities was chosen to become a main piece in the first layer of the top-down approach.

According to ideas exposed in the previous subsection, the entity id is responsible to satisfy agent body needs. These needs are called drives and together with body perceptions, that are homeostatic data and come from sensors values, permit the generation of things presentation. This concept will be explained later but is the basic data structure used in the primary process of the mental model.

Obviously, the entity Ego is the main pillar in this first layer. The Ego task is to be the psychic mediator between drives demands (id) and social rules and morality (superego). Basically, it has two functions: deal with contradictory inputs and prioritize them in order to focus on a small selection of environmental and body data and take a decision that will permit improve the agent situation according to these prioritized data. It will responsible to apply defence mechanism when it would be necessary.

As was explained, ego contains what is known as deliberative functions, that permit to take a decision. These functions have as requirements two specifications. The first requirement is to dispose of proper incoming sensor data in order to establish the current situation of the agent. The second one is to dispose of the necessary actions to be able to alter environment condition that surround the agent in the complex world.

The last entity in this first layer is Superego. The porpoise of it is evaluate the actions and desires of the id and give a feedback about if them are allow to be fulfilled or done due to it is able to estimate actions impact before to be execute in a specific situation. As a result of this task, superego can be seen as the key component of ethical reasoning.

Just to give an idea of all concepts described before, next figure shows the second layer of ARS model. This model and the loop that is followed by inputs data and the decisions are explained in the chapter 6. All the ideas develop in this subsection have a representation in one of the modules in the second layer.

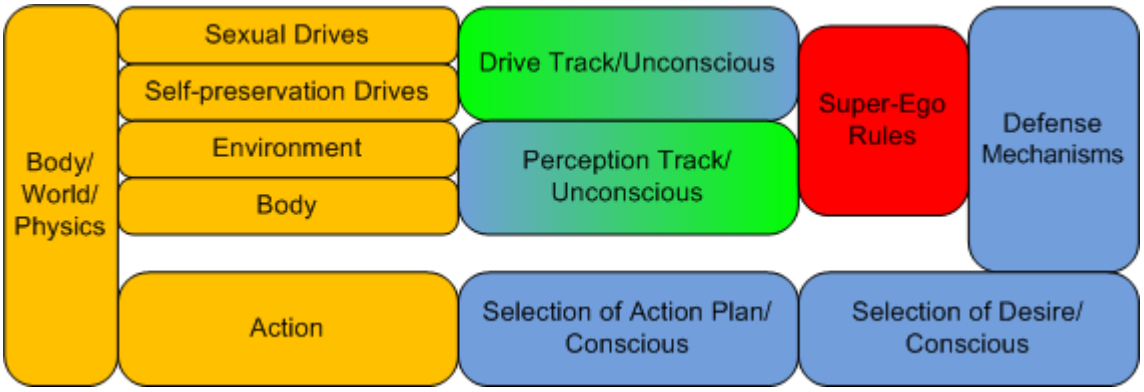


Figure 2.8: Second layer of ARS model [1]

Theoretical Approach

The primary goal of this project is to develop a more human like decision-unit that can cope with large amounts of input-data, filter out the relevant information, compare redundant sensor information and use this for more fault-tolerant and robust applications in the fields of surveillance and building-automation.

To achieve that, it has been design a model that has been reviewed and analysed many times. Besides, one feature of the model was always clear: The model operation cycle is: perception, decision and action.

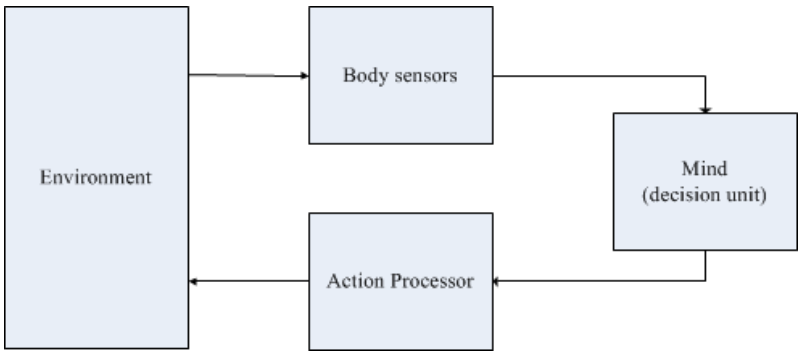


Figure 2.9: Model operation cycle

The figure 3 shows how flow data in ARS model is. Firstly, the information of environment is captured. To do it, different sensors have to map this information to be provided to ARS engine. This is the first step and it should be done in a good way. All these information recollected from the environment is transferred to Mind entity. This entity will compute, according to psychoanalytical theories, and will take a decision that will consist on executing the best action in this moment. After that, the command will deliver to Action Processor. This final entity will execute the action in order to interact with the environment.

All this process must be complemented by feedback information. It seems obvious that ARS engine cannot remain static. To avoid this situation, all information, that have been processed and analysed, have to be used to improve mental process of the embodied agent and its way to perceive the environment as well as.

Technical Approach

Before stating talking about how ARS model is designed, some concepts must be defined. The first concept is drive. Freud describes [Boe97] a drive as "psychical representative of the stimuli originating from within the organism and reaching the mind, as a measure of the demand made upon the mind for work in consequence of its connection with the body".

Some other concepts surround it. It is possible identify aim of drive as the need to remove the stimulation that causes the drive to be active. As consequence of that, it means that there is a source of drive; normally an organ can be identified as the source. Finally, the thing that allows achieving the balance again is known as object of drive. An example can clarify all these concepts. For instance the eat drive has as a source stomach fill level, it aim is nourish and its object is food. Other concept that needs a special mention is affect. The term affect shows the quantitative component of the drive. It can be defined as the drive demand intensity as well.

Once a theoretical concept of the model is establish, it is time to go deeper into the model and start to get a technical approach of it. As a first view, psychoanalysis makes a difference between two psychic encoding mechanisms: primary process and secondary process. Each of these processes permits to summarize a basic group of functionalities.

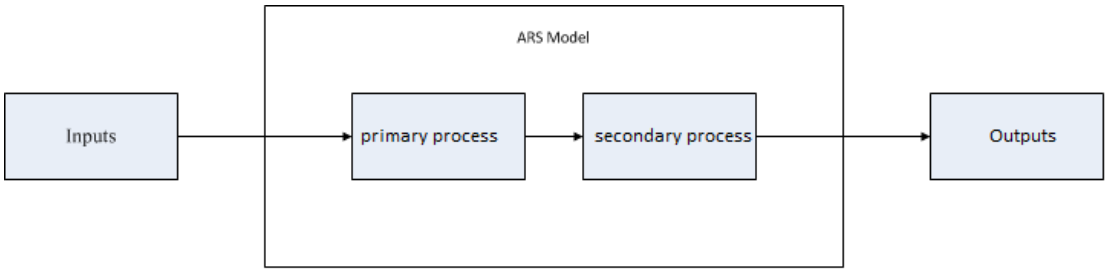


Figure 2.10: First technical approach

On one hand, primary processing can be seen as the entity Ego. It means that it involves the primitive part of the decision making. This part is responsible to generate the primitive pulse such as sexual or self-preservation drives. In particular, body internal parameters and perceptions are analysed and are converted to drive demands. So a good definition of primary process is that process that is the source of the psychic apparatus and is controlled by drives following the pleasure principle.

In this first part of the ARS model, a basic atomic data structures is defined: thing presentation. The term thing presentation can be understood as a global concept without any specification regarding the content of the information contained within each of it. However, a thing presentation [Zei10] can be the representative of a bodily memory.

On the other hand, secondary processing appears the other two Freud entities: Ego and Super-Ego. In this second part of the process, contradictions and conflicts are solved. Secondary processes cover functionalities like goal decision making, planning, and the thinking itself. Social rules are introduced and reality experience is checked due to recollect all the information to allow the decision making do its task.

Whereas thing presentations are the main content of the primary process, word presentations are the main content of the secondary process. A word presentation is the description of an object by the use of a set of symbols. It is a closed entity whose special task is to gather the associations of the object together as the complex that constituted the object identity.

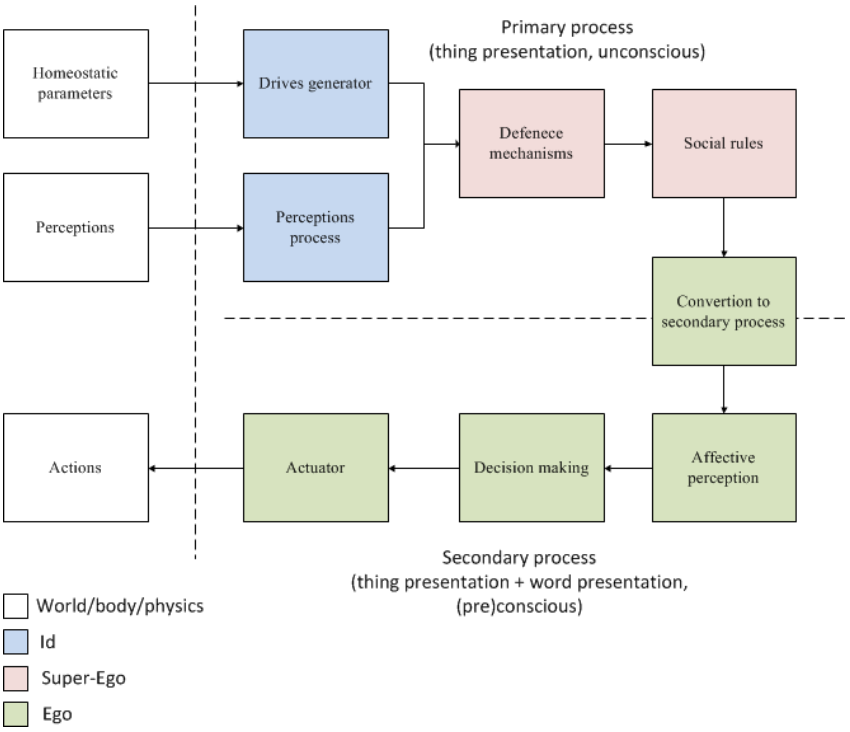


Figure 2.11: Second technical approach [Deu10, p.78]

In order to fix these two processes, a connection between thing presentations and word presentations must be done. In the beginning, a word presentation is formed out of visual images for script and print, kinaesthetic image and an image of sound that are things presentations. After this first step, word presentations are linked to objects representations, that are known as thing presentation mesh.

Functional Model

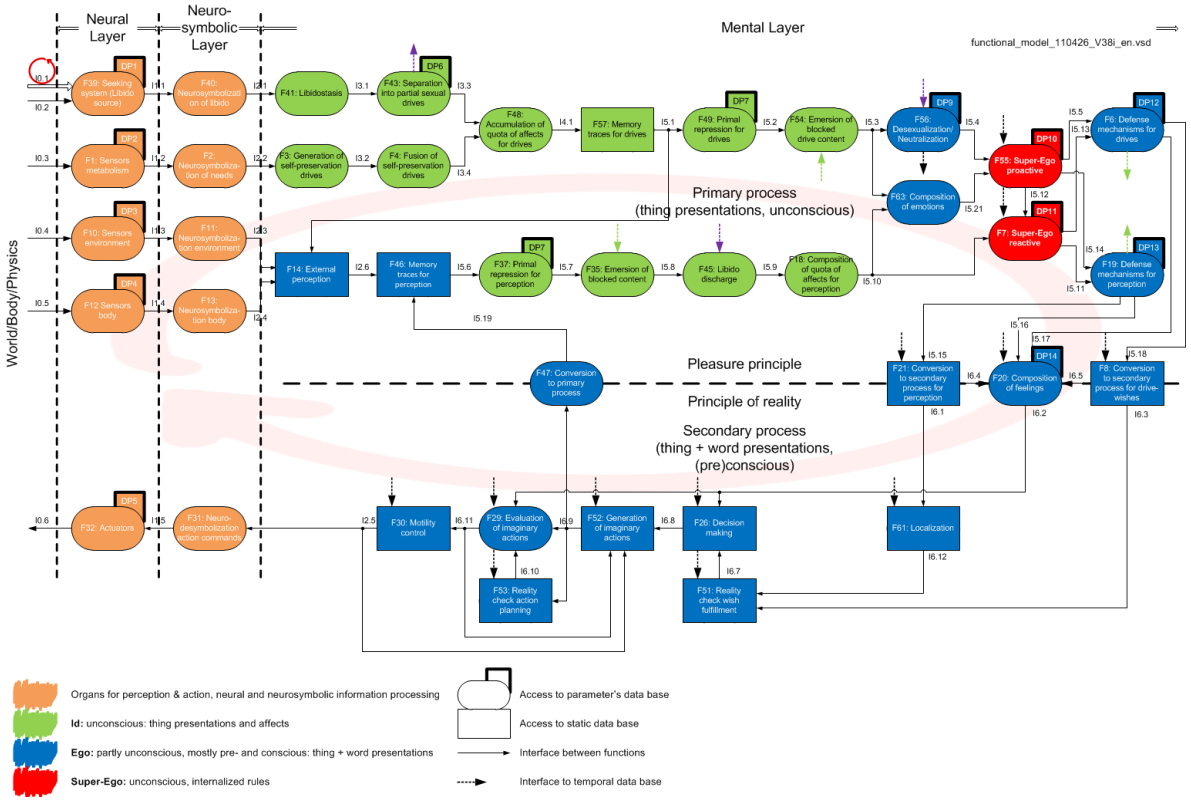


Figure 2.12: Functional ARS model [1]

The functional model depicted in Figure is built of 41 function models that some that some of them are implementations of interfaces [Deu10]. This subchapter will take a look on the main parts of the dataflow to show final implementation of ARS decision unit [Lan10]. In the next picture a picture of the complete functional model.

In the dataflow three main parts can be identified: where the inputs are recollected, where the decision making process this information and actions are decided to be done and where outputs are given. For the porpoise of this master thesis, the decision making will not be modified so it means the UT2004 bot brain will be ARS decision unit and the other two parts will be adapted using the connection interface.

INPUTS

If the inputs are analysed [Zei10], it will be possible to identify four kinds of data related to: Sexual drives, self-preservation drives, environment perception and body perception. Moreover, a data input

division can be established: drives inputs and perception inputs. The inputs, and the sensors to recollect these inputs as well, will be analysed.

Talking about drives inputs, four modules are responsible to get them: F39 (Seeking system), F40 (Neurosymbolization of libido), F1 (Sensors environment) and F2 (Neurosymbolization of needs). While F39 and F40 are related to sexual drives, F1 and F2 take care about self-preservation drives. Just to make a differentiation between these two kinds of drives: self-preservation drives are responsible for preservation of the agent own existence and the aim of sexual drives, that drive tension is known as libido, is to get as much pleasure gain as possible. It is necessary remark that sexual drives are not reproduction drives.

The libido level or, what is the same, the intensity of sexual drives are introduced toward two double values using the interfaces IO.1 and IO.2. Self-preservations drive level is introduced using the interface IO.3. In this case, the variable is a HashMap<eSensorsIntType, clsDataBase>. A hashMap permits recollect objects inside it but without order. Each object is identified using a proper identification key. In the IO.3, the identification key is a eSensorIntType and clsDataBase is the entity that store the information of each sensor. To clarified concepts, next figure shows the enum eSensorIntType used in the first ARS simulation.

```
public enum eSensorIntType {
   UNDEFINED,
   //internal system (not homeostasis)

    //homeostasis
    ENERGY_CONSUMPTION,
    HEALTH,
    STAMINA,
    STOMACH,
    TEMPERATURE,
    FASTMESSENGER,
    ENERGY,
    STOMACHTENSION,
    SLOWMESSENGER,
    HEARTBEAT,
    INTESTINEPRESSURE,
}
```

Figure 2.13: enm eSensorIntType

Related to perceptions inputs, again four modules are implemented to get them: F10 (Sensors environment), F11 (Neurosymbolitzation environment), F12 (Sensors body) and F13 (Neurosymbolization body). Using them, two kinds of inputs can be recollected: environment perception [F10, F11] and body perception [F12, F13]. In the module F10, the typical sensors are sight, hearing, smell, touch and taste. Other sensors can be used here, for instance in the first simulation a non-human sensor like radar is part of this module. In the case of environment perceptions, the interface IO.4 is used to get the proper inputs that is a HashMap<eSensorExtType, clsSensorExtern>. If interface IO.5 that is related to body perceptions is analysed, the entity that was created to store the inputs information is a HashMap<eSensorExtType, clsSensorExtern> as well. The same class is used as a key identification in both interfaces due to both kinds of inputs in fact comes from the exterior sensors. While environment inputs recollect information about the world that surrounds the agent (what

the agent sees), body inputs are related to information about the agent (where the agent is). Next figure shows the enum eSensorExtType used in the ARS simulation.

```
public enum eSensorExtType {
    UNDEFINED,
    ACCELERATION,
    ACOUSTIC,
    BUMP,
    EATABLE_AREA,
    MANIPULATE_AREA,
    OLFACTORIC,
    TACTILE,
    VISION,
    VISION_NEAR,
    VISION_MEDIUM,
    VISION_FAR,
    POSITIONCHANGE,
    RADIATION,
}
```

Figure 2.14: enm eSensorExtType

PROCESSING

As was explained in the technical approach, the processing part of the ARS model is divided in two parts: primary process and secondary process. In this subsection an idea of how the decision unit works and a description of each module will be avoided in order to do not get lost in details and focus on ideas [Zei10].

Starting from the upper part of the model, four modules can be identified: F41 (Libidostasis), F43 (Separation into partial sexual drives), F3 (Generation of self-preservation drives), F4 (Fusion of self-preservation drives). While the first two modules are responsible for sexual drives, the other two take care of self-preservation drives. The task of F41 is to recollect the incoming libido to a buffer and the total amount of buffered libido, that has aggressive and libidinous components, is transmitted to the next module. In the module F43 the aggressive and libidinous drives are split according to predefined templates and the result set consists of eight sexual drives. The modules F3 and F4 work different due to self-preservation drives has a direct mapping to inner somatic needs, so a buffer is not required. In F3, each drive is depicted into: drive source, aim of drive and drive object, and two drives are generated (the same as sexual drives): a libidinous and an aggressive one. In F4, the libidinous and aggressive drives are combined to pair of opposites.

After the drives' recollection, the concept memory traces, that define the way events are established in memory, appear in the model. Four modules processed these parameters: F48 (Accumulation of quota of affects), F57 (Memory traces for drives), F49 (Primal repression for drives), F54 (Emission of blocked drive content). Using the interfaces I3.3 and I3.4, the module F48 get all information related to drives and it task is to accumulate the amount of total stored libido, that equals the tension of the sexual drives, and quota of self-preservation drives. Then, all this information is send to F57 and it is attached to the memory traces. So in this moment of the dataflow, a thing presentation consists of drive source, aim of drive and drive object and quota of affects. In the module F49, a distinc-

tion between controllable and uncontrollable aspects of reality is made and thing presentations are organized to a scheme originating in partial-drives. They are categorized according to the concepts of oral, anal, phallic, and genital. The last of these four modules is F54 where frustrated contents are match with material representation of perceptions, if a relation is found. Thanks to that, repressed information is tried to be treated in different contexts.

Following the thread that processed the perception information, six modules can be identified: F14 (External perception), F46 (Memory traces), F37 (Primal repression for perception), F35 (Emersion of blocked content), F45 (Libido discharge) and F18 (Composition of quota of affects for perception). F14 gets data provided by sensors of environment and the body and transforms then into thing presentations, associating each other according to their temporal and spatial likeness. After that, these things presentations are associated with previously experienced and stored into memory traces. This is the task of F46. Then in F37 and in F35, the same idea as the module F49 and F54 is done, but in this case the treated information is related to perceptions instead of drives. In previous versions of the ARS functional model these four models were only two, treated at the same time drive and perception information but for clarity and simplicity of the model it was decided to split them in two flows. After these two modules, F45 compares incoming perceptions with memory and decides if they will have a libido discharge as a repercussion. The libido discharge or, what is the same; the pleasure gain is send to F18, that has a similar task than F48, as a value to the composition of the quota affect.

Once thing presentations and memory traces are defined, it is time to play with social rules, that are part of Superego. In this part of the process, four modules can be identified: F56 (Desexualitzation/Neutralization), F63 (Composition of emotions), F55 (Super-Ego proactive) and F7 (Super-Ego reactive). The task of F56 is to control the amount of neutralized energy and generated pleasure according to this energy. In F63, the level of emotions, that are a measure of the amount of pain, are specified. “The object is the most variable on instinct,” Freud writes. This is relevant especially in the sexual instincts, that only requires to sustain life, but for the pleasure to serve. So here the categorization of the libido on the source seems more important than for the object to that it can be satisfied. The outputs of these two modules are introduced to F55 using the interfaces I5.4 and I5.21. Basis of the superego is: for infringement of the organism must be punished with pain. It must also train and exercise content to be balanced in their views of the intensity of pleasure resulting from the pain, the content should be approved contrary to the rules of the superego. F55 operates independently of the current driving situation and perception, and provides general claims. The last module related to Super-ego is F7 where rules that are only accessible to functions of the Superego are used to evaluate the incoming drive demands and perceptions. Three possible decisions can be made for each incoming information: they can be passed on without any changes, they can be passed forward but certain changes have to be made, and these contents are not allowed to pass at all. If the evaluated contents qualify for one of the latter two possibilities, it means that a conflict occurs; defence mechanisms have to deal with them.

Once the modules of F7 and F55 evaluate incoming information and categorized them, F6 (Defence mechanisms for drive) and F19 (Defence mechanisms for perception) treated with no-stables situations. F6 select that drives will be categorized as conscious and that no. If a drive is not allowed to

be conscious, a defence mechanism is selected to treat it. For instance, a defence mechanism can be repress a thing presentation in the current moment or attach it to more things presentations. Analogous, The F19 realizes the same task but in this case taking care of perceptions.

Outputs of the defence mechanism modules are forwarded to three modules: F21 (Conversion to secondary process for perception), F20 (Composition of feelings) and F8 (Conversion to secondary process for drive-wishes). These modules are responsible for converting the data structures between the primary process (thing presentation) and the secondary process (word presentation). In this task, the module F47 (Conversion to primary process) participates as well. Information that comes from F19 is used by F21 to associate the thing presentation and quota of affect generated by incoming perceived symbols with the most fitting word presentations found in memory. In the case of F8, the information comes from the module F6 but the task is the same. In these module the concept drive contents are replace by drive wishes. The module F20 convines information from F6 and F19 to compose feelings. It is important to remark the qualitative quota counter-part of affect in the primary process is the affect in the secondary process and when the affect is represented by a word presentation then can become conscious. Some examples of these generated feelings are fair or joy. Finally in the module F47, word presentations are converted to things presentation to produce feedback and reduce libido tension (deleting the preconscious parts).

In the last part of the dataflow, where the decisions are taken, seven modules are implemented: F61 (Localization), F51 (Reality check wish fulfilment), F26 (Decision making), F52 (Generation of imaginary actions), F29 (Evaluation of imaginary actions), F53 (Reality checks action planning) and F30 (Motility control). In F61, all the staff is prepared for the module F51 in order to get all requirements to the decision making. It is in F51 where basic knowledge of external reality is processed. The knowledge about the reality affects the reality check (for example the idea that raw potatoes are inedible) and this knowledge exists in memory lane that is part of semantic memory. Although semantic memory saves the information we share with other members of our society, particularly with our peer group, it also collects personal objective information (date and place). In this moment, demands from reality drives and Super-Ego are evaluated to get a motive for an action and a list of produced motives is ordered according to their capacity of satisfaction. This is the task of the module F26. After decision making, F52 combines the motives and experiences to set of imaginary action that defines a sequence of actions in order to satisfy a need based on actions taken in similar situations in the past. When imaginary actions are generated it is time to evaluate them. F29 also uses the outputs of F53 that contains lexical knowledge about the world. This includes knowledge about function and attributes of objects. The result of F29 is a reduced list of imaginary actions for execution. The last module is F30. It uses drives inhibition to evaluate how the submitted action plan can be realized best. Drives inhibition leads to the possibility to perform behaviour in rehearsal.

OUTPUTS

Finally, the last part of the model is related to outputs. The modules F32 (Actuators) and F31 (Neurodesymbolitzation action commands) are responsible to give them towards interface IO.6. The output variable, that comes from F32, is an ArrayList<clsActionCommand>. This clsActionCommand is an abstract class permits recollect the actions that have been selected to be done by the decision unit.

Each action implements the class clsActionComand and all possible action that can be done by the agent must be declared. All the actions and the action parameters as well are represented using XML. For instance, when the information about the action of the class clsActionAttackBitte is wanted to be extracted, this is the result: <AttackBite> 0.7 </AttackBite>, where the value between tags means the strength of the bite. Moreover, a sequence can be created as a combination of simple actions in the class clsActionSequenceFactory. For example, the salsa sequence, that the salsas dance, is a combination of simple sequences: move_forward and turn_right.

ARSIN: First ARS simulation

All of this humanlike thinker architecture does not have sense without an embodied agent and he does not have sense without an environment to be able to act as human. In this subsection a description of the environment and the agent will be given [Deu11]. It has an importance due to it was a precedent to be able to insert inputs and received outputs from the entire ARS engine. Although the GUI of the simulation will be not analysed in this master thesis, it results helpful when the embodied agent parameters are wanted to be visualized.

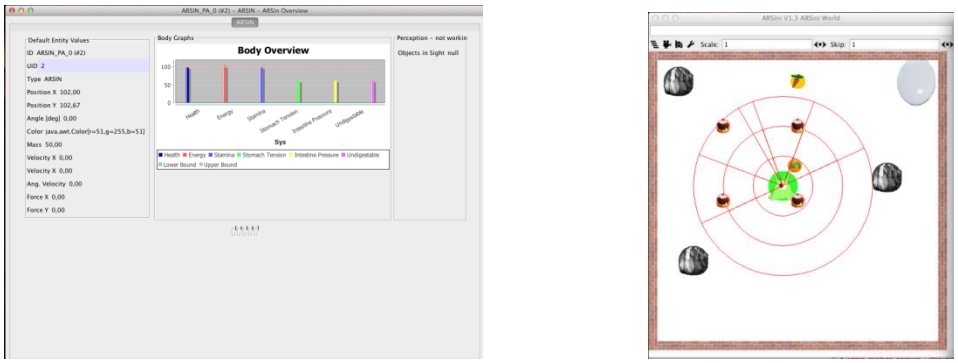


Figure 2.15: Screenshots of ARSIN

ENVIRONMENT

Once ARS decision unit had been implemented in a stable version, an environment was designed by ARS research group. Although it was a simple world, it permits to test if the model that was built was enough revised. It was based on a 2D world where different kinds of objects could be founded. These objects give the opportunity to the agent to satisfy its drives. The next figure shows a screen-shot of one simulation.

Every object was created taking care of that at least satisfice one drive. For instance in the figure 4 four objects can be identified: stone, cake, carrot and toilette. Moreover, two more objects must be considerate: wall and ARS agent. The first four objects that have been cited are created as objects of drives. Next table shows drives that must be satisfied in the embodied agent.

Drive	Source	Aim (libidinous)	Aim (aggressive)	Object
Eat	Blood sugar level	Nourish	Bite	Cake
Consume	Oral mucosa	Stimulation with object at source	Bite	Pacifier
Defecate	Rectal mucosa	Repression	Expulsion	Pile of faeces
Urinate	Urethral	Retention	Squirt out	Urine
Genital sexuality	Genitals	Excite	To suffocate	Opposite sex
Sleep	Stamina level	Relax	Sleep	The own body
Breathe	Trachea	Breathe	Annihilate	Air

Table 2.1: Drive of the ARSIN

In order to implement these objects into ARS engine the following architecture was designed. A main class was created called clsEntity. In this Java class generic variables are defined such as string identification or Entity type. There are two classes that implements clsEntity: clsStationary and cls-Mobile. In the case of clsMobile, velocity in cartesian coordinates are defined as properties. From clsMobile two classes are implemented: clsAnimate and clsInanimate. This was the classification and, as consequence, the architecture established for the ARSIN objects.

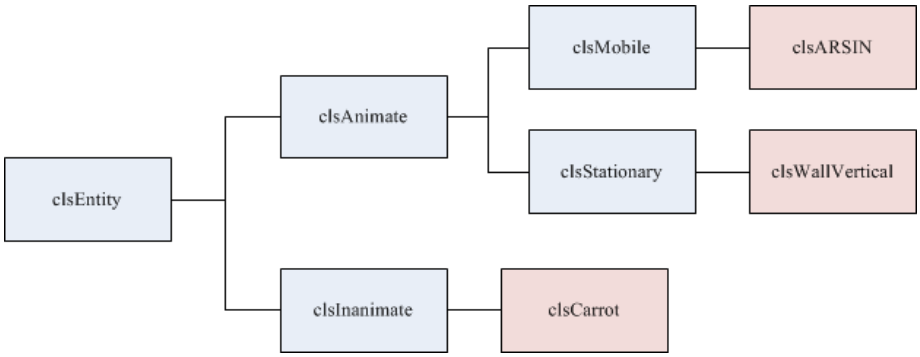


Figure 2.16: Taxonomy of ARS objects

Due to the need of the embodied agent to recognize when it see them, some properties must be defined for each object. As main properties, colour and shape were defined. Also other characteristics were assigned to complement the object information. It plays an important role when UT2004 objects are wanted to introduce in ARS engine as it is explained in the next section of this chapter. Next table shows the properties of some defined objects.

Object	Colour	Shape	Body	Mobile
Wall	Black	Rectangle	Simple	No
Stone	Grey	Circle	Simple	Yes
Cake	Pink	Circle	Meat	Yes
Plant	Orange	Circle	Meat	Yes
Carrot	Orange	Circle	Meat	Yes
ARSIN	Green	Circle	Complex	Yes

Table 2.2: Objects in the environment

EMBODIED AGENT

When the class clsARSIN was designed, two restrictions where defined by definition. Firstly, it must be able to get internal information and, at the same time, external information. To do that task, the embodied agent must dispose of several systems and sensors. Secondly, the ability to reproduce human acts must be present as well.

Related to Internal sensors a class called clsInternalSystem was created to deal with body internal concepts like health or communication between internal systems. From this class, other entities were implemented to take care of a specific field. Although each module works apparently independent

from the others, parameters are exchanged among them to get a feedback of each one. The internal parameters used are shown in the next figure.

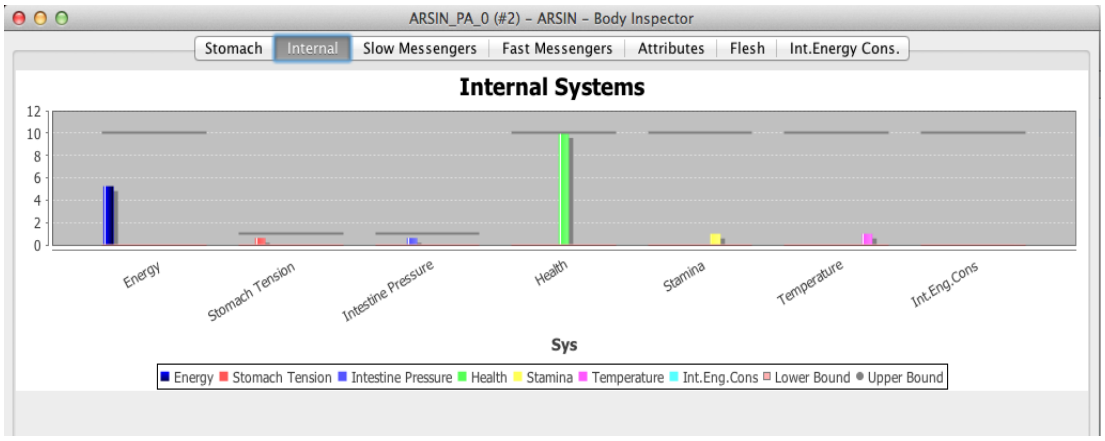


Figure 2.17: ARSIN internal parameters screenshot

A value, that can be identified, is stomach tension. It is used to compute using tension the consumed food [Deu11] and, once it is done, the potentially available energy. Then energy consumption make an estimation of quantify the energy needed to perform all actions. Another concept related to energy is Stamina. If an action is done, apart from energy consumption also certain stamina is required. This parameter is related to self-preservation drives due to if stamina level is not enough high to do the action the embodied agent needs to rest. However, energy level is difficult to consume and recuperate. Another internal parameter was temperature that controls the embodied agent body temperature. It can change depends on environment conditions. Intestine pressure is also related to self-preservation drives due to it is that controls the need to excrete.

The last parameter that is represented is health. Basically two situations can produce a reduction of health: if the embodied agent crashes against the wall or if the level of energy is too low. This parameter will be used as a main internal body value to connect UT2004 environment with ARS decision unit.

Making a last appreciation to the concept energy, there are eight parameters that decide the energy level. Objects like cake or plant must be eaten to increment these parameters otherwise their values would be decreased. The next table shows a list of objects and what they contain.

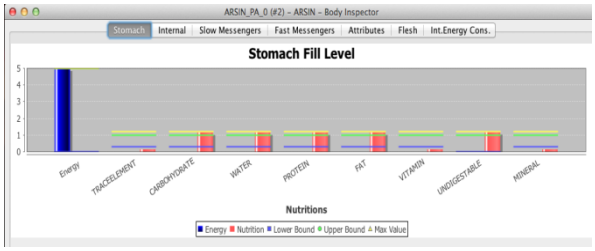


Figure 2.18: ARSIN stomach parameters screenshot

	Undigestable	Water	Fat	Carbohydrate	Protein	Vitamin
Cake	11.1%	5.6%	27.8%	27.8%	27.8%	0.0%
Plant	70.4%	14.1%	7.0%	7.0%	0.7%	0.7%
Carrot	11.1%	55.6%	0.0%	11.1%	0.0%	22.2%
ARSIN	16.0%	40.0%	20.0%	20.0%	2.0%	2.0%

Table 2.3: Objects nutrition

The next aspect must be treated is the system of visual sensors. As was commented before, ARSIN environment is based on 2D world. In the model chapter it will be explain how this conversion 3D-2D because of the connection between UT2004 and ARS decision unit. The embodied agent vision is designed using a field-division concept. The space is divided using two classifications that combined give the 15 vision regions.

- The first classification divides the camp of vision among near, medium and far. Everything that is outside of the far limit is assigned to undefined value.

- The second division divides agent vision among left, medium_left, center, medium_right and right.

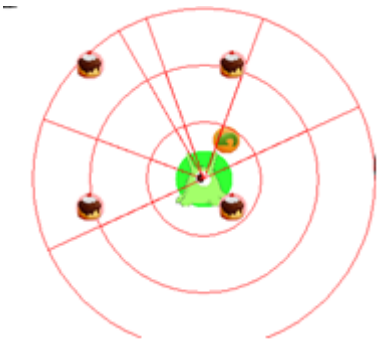


Figure 2.19: Vision of ARS agent

These divisions imply that the embodied agent doesn't know exactly the position of a cake for example (talking about cardinal coordinates). It only knows that it see a cake in the [LEFT, FAR] field.

2.2.2 Unreal Tournament 2004

Here, an explanation of the computer game used in the course of this master thesis is done in order to show how UT2004 works, how to adapt it to our porpoise and advantages and disadvantages of it.

Unreal Movements

Unreal Tournament 2004 is an open source game that has been developed by Epic Games with the collaboration of Digital Extremes. It is a first-person shooter computer game and it has a great recognition among computer game players due to this is the third version of Unreal Tournament serial games.

UT2004 provides a huge sort of options related to movements, actions, sensors acknowledge and more aspects [Ut04]. In this way, it is possible to find whatever it is necessary to full field ARS model and, at the same time, it is easy to adapt ARS outputs to UT2004 environment.

Although it could seem one bot follows complex actions, that can be to flee or to follow, all of them are a combination of simple actions as move forward or jump. On the following table it is shown a list of the simple actions that will be used to execute the outputs from ARS engine.

Simple Action
Move forward
Turn right
Turn left
Shoot
Jump
Crouch
Choose weapon
Drop

Table 2.4: UT2004 simple actions

As it was said before, simple actions listed are the tools necessary in order to create complex action as can be flee. For instance, in the case of flee, the simple actions move forward, turn right, turn left, jump as well as the information that body sensors will give (sight) have to be combined in order to achieve to lose the bot enemy.

Complex Action
Flee
Follow enemy
Pick up an object
Kill enemy
Navigate

Table 2.5: UT2004 complex actions

Apart from simple and, as a consequence, complex actions, on the environment of UT2004 exists a group of objects that provide different aspects to the embodied agent. There are objects to satisfice different needs of UT2004 Bot. For example, there are objects for health to satisfice self-preservation drives. In total there are 25 different objects that are shown on the following table.

UT2004 object
Health pack
Armour
Adrenaline
10 types of weapons
11 types of ammo

Table 2.6: UT2004 objects

Unreal Environment

On the programmer side, UT2004 gives a huge sort of facilities when someone tries to work with it. Firstly, Pressing ”alt” + “h” the main menu is enabled making possible be able to watch some graphic stuff or technical information that is enable to be seen. All of this information doesn’t have any effect on the normal course of the game. The Figure 6 shows this main menu and that letter has to be press to select one of them.

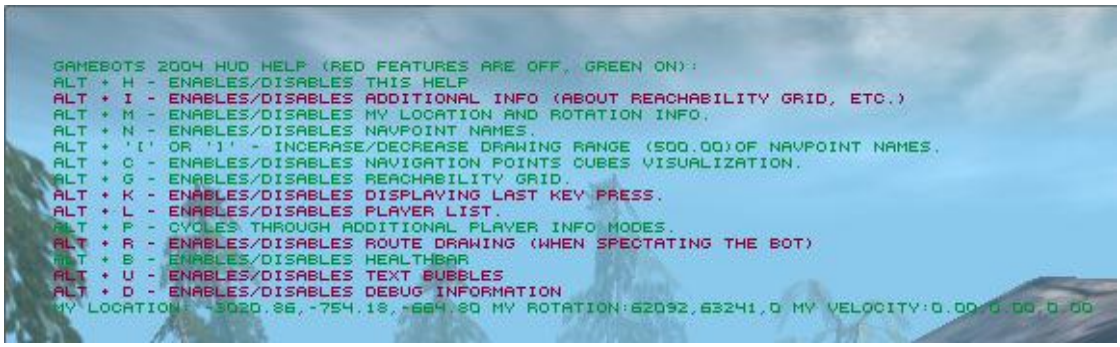


Figure 2.20: Screenshot main menu

In special in this menu the most useful is the option to enable or disable the reachability grid. It means activate the yellow line (Figure 7) that represents the possible trajectories of an embodied agent. This agent follow this lines because of he has the desire to explore the map. It exists the possibility to not follow these patterns making the experiment more accurate.

Figure 2.21: Paths screenshot

Figure 2.22: Navigation points screenshot

The reason of being of navigation paths is to connect all navigation points among them. As it can be seen in figure 8, a navigation point can have two meaning. The first one is that it can be a simple fixed point of the map, is the case of the picture. The reason of this option is to achieve the map as mapped as possible.

The second option is to be a fixed point where the embodied agent can find weapons or the ammo. Some module of the bot must take care of the memory that will be used to store the location of this point to go again in this point if it is necessary. One interesting thing is every navigation point has its own identification. It permits that the embodied agent can be able to remember every point that has been patted.

Unreal Advantages and Disadvantages

If the game role is analyzed in order to see advantages and disadvantages of it, some aspects must be remarked due to will be important to be able to do the work proposed until now. Starting from advantages, the first one could be UT2004 is a complex world that provides a real-time dynamic multi-agent environment and, not only that, also a large variety of objects to be percept and actions to be done by the agent. Secondly, UT2004 is an open-code (except the graphical interface), that is not the same as open source. It gives the possibility to create personalized maps and to modify the UT bot behavior using Pogamut tool, explained in the section 2.2.3.

On the other hand talking about disadvantages, there is a big difference between ARSIN environment and UT2004 world. Secondly, there is a difference between UT2004 and ARS process speed. In other words, ARS is too slow to provide actions so UT2004 bot has to wait for them. Finally, new action such as flee and new objects like weapons must to be created.

Example of an Unreal Implemented Bot

Some bots were implemented before this work started to get shape. Just to give an example of them and to show that there are a lot of projects opened and researcher groups working using this environment around the world.

Q-LEARNING

Some years ago, the challenging task of defining human-like behavior for non-players agents in real-time A. Goyal and P. Pasquier chose a Q-learning approach. To create these bots and run into UT2004, they were using the pogamut tool as it is done in this work. As will be explained in the next section, it simplifies the entire embodied agent physical part and gives the possibility to design and debug agent behavior.

First of all, before talk about the concept Q-learning, it can be more proper start talking about Reinforcement Learning (RL). As it is defined in the document [Goy11]:

> "RL is a machine learning technique of mapping situations to actions so as to maximize a numerical reward signal. In RL, the computer is not told that actions to take, but must discover that actions yield the most reward by trial-and-error interactions with the environment."

So, among RL technique there is Q-Learning that makes a math between state and actions pairs after the action has been done and the reward achieved. These pairs are called Q-values and are computed basically using four parameters: learning rate (alfa), the reward (r), action taken (a) and state where the agents is in the moment to decide the action (s).

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma Q(s') - Q(s,a))$$

To give an implementation to this approach, two behavior trees were designed. The first one was for navigation mechanic and the other one for shooting mechanic. In the first case, the task of the BT is gives the functionality to navigate looking for health, player, ammo and weapons. In the second case, the task of the BT is gives functionality to act when a player is seen.

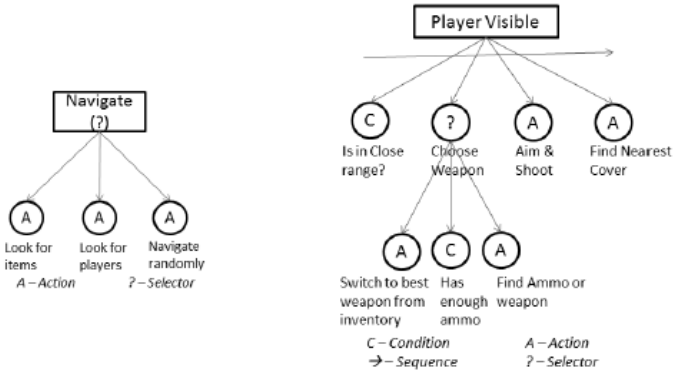


Figure 2.23: Design of BTs [Goy11]

The results presented related to using an RL Q-learning algorithm (using BT to implement them) was that them give the required resources to embodied agents the ability of adaption in a real-time dynamic environment such as UT2004. Also, the feature of using two BT at the same time gives multi-behaviors agent that is essential in this kind of environment in order to be able to survive in them.

2.2.3 Pogamut

Due to UT2004 uses its own script language, an external platform is required in order to get a proper communication between the computer game and the IDE used to develop the ARS model. Fortunatelly, not only it exists, moreover it is free. This platform is known as *Pogamut*.

Pogamut Architecture

Because of the necessity of implement Bot in a complex world, it has been chosen the Unreal Tournament 2004 to be this world. To achieve insert a personalized embodied agent, the essential tool is Pogamut. As says Pogamut documentation:

> "Pogamut is a Java middleware that enables controlling virtual agents in multiple environments provided by game engines. "

The main goal of Pogamut is to enable user to focus his efforts on behaviour aspects. It means that user do not be worried to the physical part of agent creation [Gem10]. It give all is necessary to create the embodied agent in Unreal Tournament 2004. Practically, all actions in the environment can be performed by one or two commands.

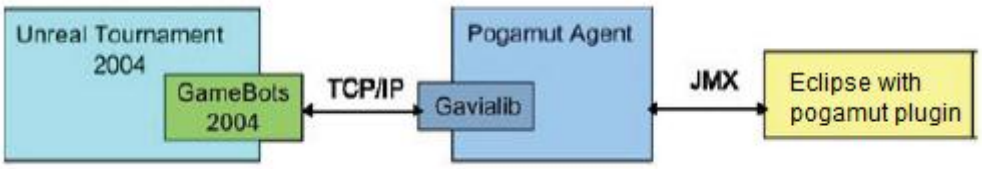


Figure 2.24: Pogamut environment [2]

Pogamut 3 covers four main aspects[Gem10]; firstly, it provides connection to the UT2004 environment. Secondly, it provides an integrated development environment and the possibility to debug it. Thirdly, all the staff related to sensors, path algorithms and weapon handling (including shooting behaviour). Finally, a complete support when running experiments wanted to be done.

As can be seen in the figure 2.18, Pogamut agent is the middle step between Eclipse and Unreal Tournament 2004. To do that, it uses a library that is called Gavialib. The function of this library is translating UnrealScript to Java, or from Java to UnrealScript. It permits to connect agents to almost any virtual environment.

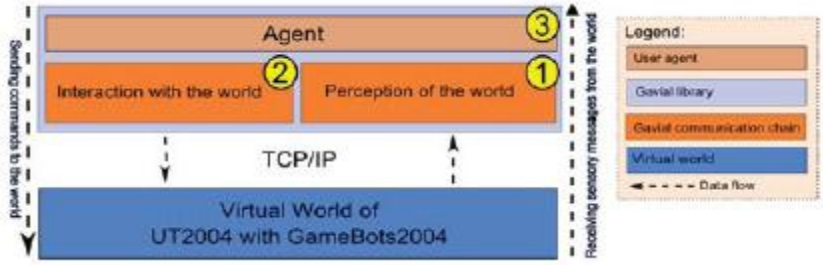


Figure 2.25: High level Gavialib architecture [2]

Another important entity is GameBots 2004. This entity is charged to export and import information of the game with UnrealScript language using TCP/IP protocol. Finally the connection between Pogamut Agent and Eclipse is made through a plugin and using JMX language.

Bot Architecture

Working with Pogamut environment facilitates the programmer work. Apart from the library GaviaLib, this provides an endless of methods to get the required information or to make with the embodied agent what is necessary in that moment, this environment brings several functions that each one has been developed to realize a specific task. These methods are called sequentially as Figure 6 shows [2]. It is important to note that additional methods can be declared but they will only have a local scope.

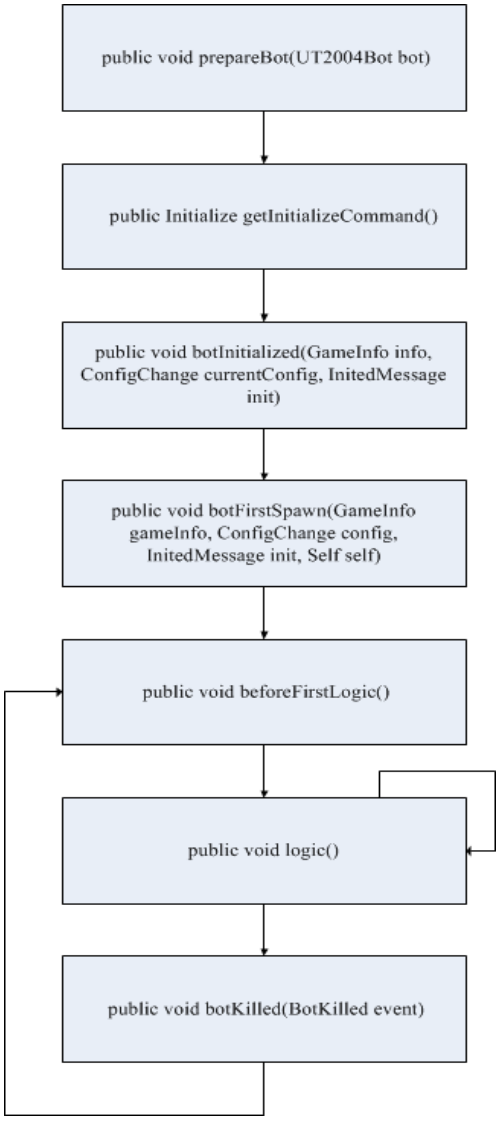


Figure 2.26: Pogamut Bot structure [2]

In total there are six methods. All of them are designed to have a special task on the course of a game. Firstly, the method prepareBot() is call in order to have the necessary information for the proper behaviour of the bot. Inside this function all necessary variables are initialized, before the bot actually receives anything from the environment. The second function that is called is getInitializeCommand(). It permits to modify initializing command of the bot. For instance, it is possible to set its name or skin. BotInitialized is called when the bot received INITED message from the server. This means that INIT command succeeded and after that, Handshake with GameBots2004 is over – bot has information about the map in its world view. Many agent modules are usable. This does not mean the bot is already spawned in the environment. The forth method is botFirstSpawn() that is called when the bot is spawned in the game for the first time. This means that the bot graphical representation is visible in the game and Selfobject was received from the environment holding information about bot location and current state. This is the last place to do some preparations be-

fore logic() method. This is the most important function in a Pogamut bot structure. It must contain all the logical processes and the behaviour that will be used to manage bot. Every second it is called four times so it means that the programmer should take care about how long the logical part of the bot is. Although only 25 milliseconds is the time that is disposed as a default, some stuff can be done; for instance if more time is required a possible solution is to go inside the implementation only in the even calls of the function. Finally, the last method is BotKilled() and as its name indicates it will contain all the code that will be executed when the bot will be killed. After that, the bot is spawned again and the logic() method is called as it was described before.

3. Model and Concepts

As it was described between the Introduction (see chapter 1) and the State of the art and related work (see chapter 2), the challenge of communicate this two environments, that are cratered independent, will be satisfied using an interface that will play the role of a bridge. The main task of this interface will be to generate the adequate inputs for ARS engine from all the sensors system of UT2004 and then be able to translate the outputs form ARS engine to generate the correct inputs for the UT2004 embodied agent.

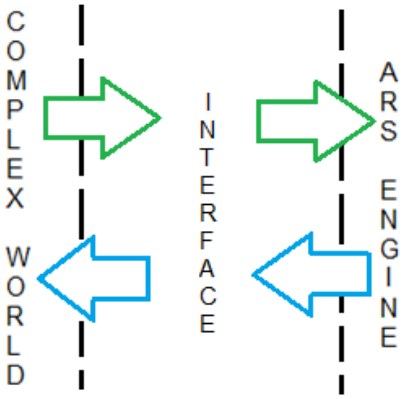


Figure 3.1: General scheme

3.1 Discussion of Architecture

A first dilemma was analyzed when the model was though for first time. As said before, two entities must be connected. It could seem that no problem should appear but the situation is two threads are working apparently independent. In one side it is the embodied agent that is living in the UT2004 and on the other side it is the ARS decision unit that is taken decisions based on the information that is provided by sensors. In order to give a solution to this problem two possible situations have been analysed.

3.1.1 Situation 1: Independency Between Both Sides

This first possibility consists on the Pogamut bot and the ARS engine work independently. It permits both sides do not take care about what is going on this two environments in the sense of the data that will come from embodied agent sensors will be introduced in a buffer. This entity will store these inputs and will wait until the decision unit will be ready to process them. It will work in the same way with the outputs that will be provided by the decision unit. The figure 8 exemplifies this behavior.

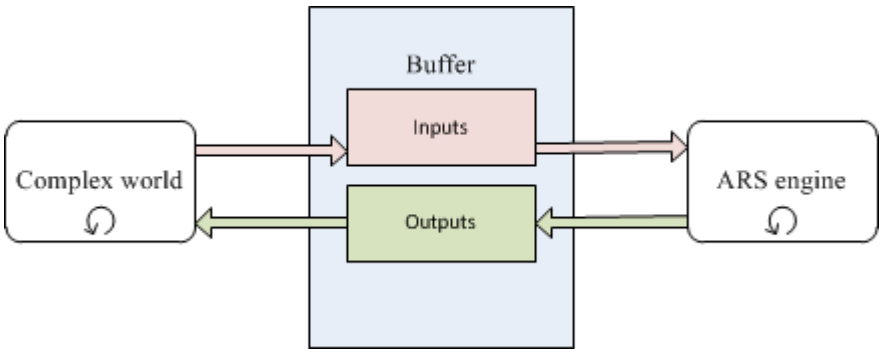


Figure 3.2: Independent model

The advantage of using this implementation is that the data is stored in one buffer, depending where it comes from. Although it permits both environment works as independent entities, it requires that both sides should have the same speed process. For instance it could be possible be in front of the situation that the input buffer is filled faster than the output buffer is emptied. In this case two problems can appear that could block all the process. Firstly, the embodied agent would receive late the actions to be done. So the behavior of the agent would be not the proper. Secondly, the input buffer could be collapsed and, as a consequence, all the environment too.

3.1.2 Situation 2: Master-Slave Model

In order to not get block the entire environment, another architecture was designed. It consists on a master-slave model where the embodied agent plays the role of master and the ARS engine acts as a slave. It means that the embodied agent will provide all data to the interface and it will wait until the interface will give the actions to be done. At the same time, the ARS engine will only be executed when the embodied agent will need to realize an action.

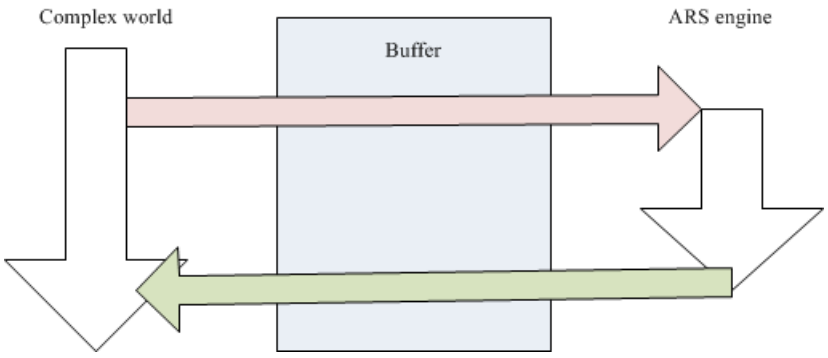


Figure 3.3: Master-slave model

Thanks to this architecture the potential problem of collapsing the entire environment is solved. The embodied agent will receive the proper actions in real time, it means that no will be introduced and, as a consequence, once per loop. It is for this reason that the architecture that has been chosen was the Master-slave model. The only inconvenient of this model is the ARS engine should be as fast as possible in the sense that the embodied agent cannot be waiting for a long period of time. But this is not a problem because of process time of ARS engine.

3.2 Environment Interaction

As said before, the interface must be able to connect both parts. To do that, the complex world will give all the inputs in order to insert the proper data into the ARS engine. Here is when Pogamut starts to work in order to facilitate the work of this task. Although Pogamut give an incredible help, it is necessary to take a look on the documentation [2]. There, it is possible to find the entire variable that will let extract all data from there.

From the UT2004 two kinds of information must be extracted. First, all information related to environment perceptions must be checked. Second, body perceptions must be processed. Basically, this is the information that the ARS engine needs in order to be ready to start the decision unit. In the next sections it will show how and from where it will be extracted.

3.2.1 Requirements for a Complex World

Unreal Tournament 2004 [3] plays an important role in this master thesis. It has been the first time that ARS model has used in an external environment. Although somebody could think that any shooter game would be appropriate, this is false. Some requirements have to be followed due to it has to be connected with ARS implementation. In the next subchapter this requirements will be analysed.

HABITABLE WORLD

The first condition that has to be accomplished is that the complex world must be habitable. It means this world must be able to receive an embodied agent and at the same time give to him the resources to interact with it in a human way. For instance, the agent should be able to do physical aspects as run, jump, turn and psychic aspects as be conscious where it is and what it sees.

APPROPRIATE INPUTS

The second condition is that the complex world must be able to generate appropriate inputs in order to guarantee a good behaviour of psychoanalytic model. ARS model needs different kinds of inputs. It is possible to split this inputs between internal and external inputs.

ACHIEVABLE OUTPUTS

As well as inputs must to be adequate, outputs should be understood for a third entity, in this case, Unreal Tournament 2004. To achieve a well connection between ARS and UT2004, an interface should be developed. This interface will be created in the course of this master thesis.

CAPACITY OF GENERATING PSYCHOLOGICAL REASONING

The embodied agent has to be able of feel drives and motivations. Both are essential if an ARS implementation in an external world is wanted to do. A clear example of a drive could be the necessity of get a health pack when it has been injured. As motivations it could be understood the desired to recollect all the weapons and armour on the map. These two concepts will be mentioned in the following sections.

RECOGNIZABLE OBJECTS

All the objects of the world should be mapped into an ARS objects. In other words, features from different items of the environment should be represented as entities on ARS memory. Otherwise ARS engine should not identify what it needs to satisfy a desire or a motivation. In this case, another interface will be created in order to translate objects that have been seen into few features that must be known by ARS engine.

3.2.2 Environmental Perceptions

First of all it should be defined what is understood for environment perceptions. This kind of perceptions includes everything related to the environment, for instance, know if the embodied agent is in front of a wall so in this case and with the proper information, the ARS decision unit should decide to turn in order to dodge it. Another example of environment perceptions could be to know if

there are item to be pick up an in this other case the decision unit should decide if pick them up or not, depending of the needs of this moment.

World

The most representative variable of the environment perceptions is *World*. In it is possible to find different methods that will gave the needed information. In the next figure, it will be shown the methods that will be used in this world even though there are more methods than them.

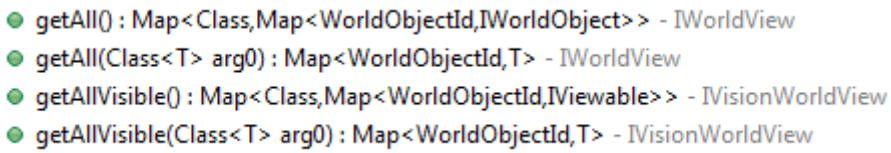


Figure 3.4: Methods of World

These functions allow getting the objects that are in the map. Using the "getAll" methods it is retuned all of the items that are on the map while using "getAllVisible" it is retuned all the items that are visible for the embodied agent. So before every loop of the ARS Decision Unit the objects that are seen by the embodied agent must be check and the necessity of them must be analysed but this work it will be explain later due to is ARS Decision Unit task. Moreover, this list of objects must be proceeded in order to create ARS engine inputs. This will be a task for the interface. It is also possible get a list of one particular object, that must be indicate as a parameter using the second and the forth method.

All of the objects received from the Unreal Tournament 2004 are instances of the class *Item*. This class provides more than twenty methods but for this work only that are shown in the figure 13. In the chapter 6.3 Interface, the explanation of how the information received from these two methods are preceded will be explained in detail.

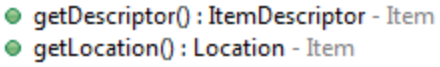


Figure 3.5: Methods of Item

Ray Castings

Interacting with the complex world it is more than see and recognize the objects that the embodied agent finds in her way. Other requirements exist when a navigation system is wanted to implement in a proper way. One of these requirements is the ability of the agent to be able to recognize when a wall will become an obstacle for its path and where it will be. To do that, Pogamut facilities ray castings sensors that will allow the agent be able to avoid walls while it is detecting them. In the next figure, ray castings sensors are shown.

Figure 3.6: Ray casting sensors

3.2.3 Body Perceptions

Moving to body perceptions, the main variable that controls everything related to, for example, where the bot is looking or that is armour level is info. Of course environment perceptions have a huge impact in front of body perceptions due to environment perceptions allow the bot interact inside a complex world giving the possibility to develop itself in it, but also a treatment of body data must be done. Body perceptions give all data about external body, for instance that is the health level or how many weapons the embodied agent has and what much ammo it has for each of them.

Info

As was said, info is the most representative variable when body perceptions of the embodied agent wanted to be known. For that reason, next figure shows info methods that are used in this work, and an explanation of them and how the interface call them in order to get the proper information.

- getArmor() : Integer - AgentInfo
- getCurrentAmmo() : Integer - AgentInfo
- getCurrentSecondaryAmmo() : Integer - AgentInfo
- getCurrentWeapon() : UnrealId - AgentInfo
- getCurrentWeaponName() : String - AgentInfo
- getCurrentWeaponType() : ItemType - AgentInfo
- getHealth() : Integer - AgentInfo
- getLocation() : Location - AgentInfo

Figure 3.7: Methods of Info

In all information related to body perceptions, four main data must be known: armour level, ammo level, health level and location of the embodied agent in the complex world. All the weapons'' and armour information will be responsible to active the correspondent drive to pick up weapons and ammo in order to fulfilled the "selfish" motivation. When an enemy hurts the agent, his health level decreases some value. Health level is related with the drive nourish that permits to generate the need to regenerated agent body. The last information that must to be identified is the location of the agent. Apart from the fact that it locates the agent in the map and, as it will be explained later, plays a main role in the conversion from UT2004 vision to ARS vision system.

3.3 Adaptation of ARS Model Into UT2004

As it has been identified, using a few words in theoretical approach as a first idea, are two main components: perception and decision making. Surrounding perception there is a group of sensors to get the information that play as the necessary interface between an agent and its environment. In the case of decision making, algorithms and internal parameters have the task of realizing the connection between perceptual interface and the action interface.

If perception is analysed carefully a group of neural connections and converts data from the human sensory organs into information as can be internal parameters, objects that are being seen by the agent, characters, images, voice or colours. In the course of this work, an interface will be implemented to adapt the actual way to parameterize data.

It seems obvious that a bot does not have parameters such as the quantity of mineral or carbohydrate that the embodied agent has. In the case of UT2004 main internal parameter is the health. For this reason, the interface will have the task of associate the embodied agent health value with the ARS internal parameter energy value. To do that an algorithm will be created to adapt the eight ARS parameters (carbohydrate, fat, vitamin, mineral, protein, trace element, indigestible and water) in order to make equal the values UT2004 health and ARS energy. Next figure illustrates this task.

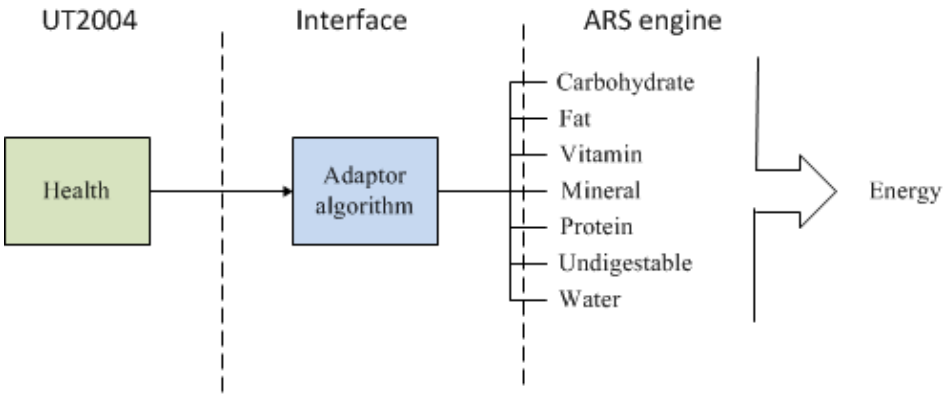


Figure 3.8: Adaptation of body perceptions

Moving on to the concept making decision, the first thing that is needed to achieve the goals of a certain agent is information about the environment. In order to get this information two properties are associated to each object: shape and colour due to them are unique for each object, so these to features describe perfectly each object. These properties will be used for the decision making to associate them to a particular object and prioritize them according to actual needs of the agent. Next table shows some examples of these objects and the main properties that they should have.

OBJECT	SHAPE	COLOUR (HEX)
Assault rifle (Weapon)	Square	00FFFF
Assault rifle (Ammo)	Circle	00FFFF
Health pack	Square	228B22

Table 3.1: New UT2004 objects

As can be imagined, objects such as plant or cake cannot be founded in the UT2004 environment. So, two possibilities were studied on the very beginning of this work. The first option was mapping UT2004 objects with ARS objects. For instance, a possible map is between health pack entity (UT2004) and cake entity (ARS). Although this option would permit an easier connection, an internal problem would appear due to both entities have not the same properties and as a consequence they do not have the same expected affect.

For this reason, the second studied possibility was crate all the object that can be found in a UT2004 map as ARS entities. This possibility was the selected option. This scenario permits to achieve a better adaptation between both environments due to UT2004 basic information is included in ARS engine in order to get a better behaviour of the whole engine. The next figure illustrates the interface task related to adapt objects from UT2004 environment to ARS engine.

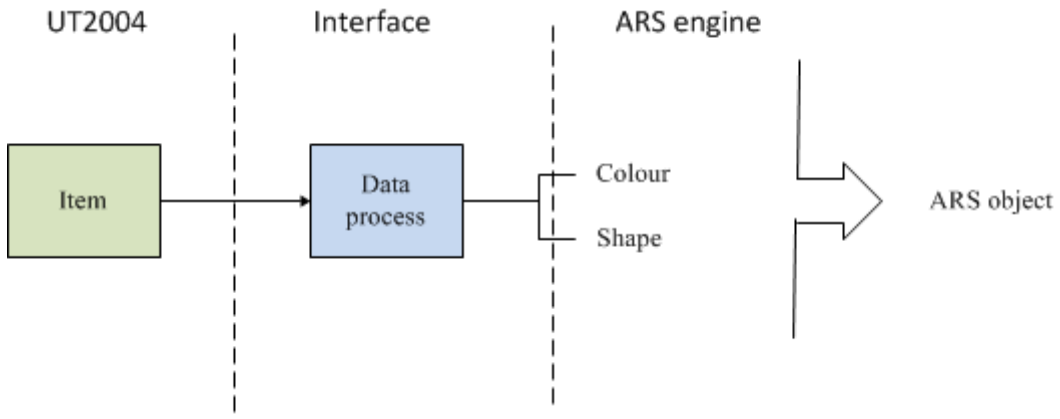


Figure 3.9: Adaptation of environment perceptions

Finally, the task to export the ARS model into UT2004 environment involves the adaptation of the psychological motivations, in other words; how the current implemented drives will be match with the new required drives. It will be necessary to find common aspects between ARSIN actions that are already implemented with UT2004 required actions. Of course, it will be cases that new drives will be essential to design.

3.4 Interface Model

Once the discussion of the interface architecture is done and the decision of how it must interact with the environment and how it must be adapted with the ARS model is taken, it is time to explain how this interface is developed.

3.4.1 Interface Task

In this part of the document the task of the interface should be more or less clear, at least a general idea of it. This is to connect the ARS decision unit to a complex world that will permit export ARS engine in a commercial application. At the moment only primitive simulation are done in order to test this psychoanalytical model and it is time to decide if ARS model has achieved the required mature to be integrated it in an external complex environment.

To do that, the interface should take care about all information that is required when a loop in the ARS decision unit is computed. It means that using the ARS input interface, that has been implemented for the occasion, must receive what it expect to receive that of course it is not the same as the complex world provides. The opposite situation, that is what information the complex world expects to receive, must be solved also.

3.4.2 Interface Architecture

From the very beginning, the main preoccupation was how to implement the interface as much adequate as possible. As was explained during the course of this work, an objective of this project was to achieve a connection between ARS engine and UT2004 that would be able to reuse in other application. Of course, each environment requires a specific interface but a general approach can be given. At least it can be a first example of how to do it for other projects that will want to use ARS.

In order to full field this objective, the interface have been implemented following a simple rule: the whole interface must be contracted in different layers that will give a clear vision of it. It means that the embodied agent will only be able to get a communication with the first layer of the interface and

then it will be responsible to send the embodied agent information among the lowest layers. The next figure gives an idea of this concept in the case of inputs.

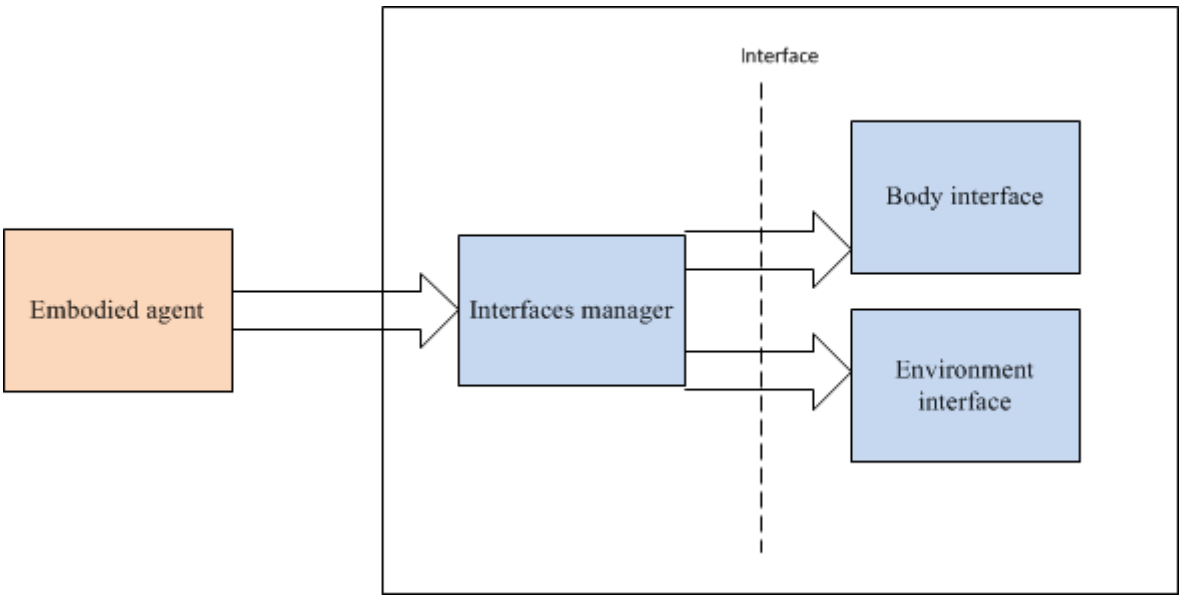


Figure 3.10: Inputs interface architecture

Inputs interface consists of two main layers. The first layer is composed by Interface manager. It plays a main role inside the interface due to it will be the responsible to distribute the sensors information, that comes from the embodied agent to the second layer. This layer contains two entities: body interface and environment interface. Both are the responsible to introduce the proper data into ARS decision unit.

4. Implementation

From here, a difference must be settled between what is wanted to do and how it is done. Here, an explanation of how the interface has been implemented will be developed and also the new entities required in ARS in order to provide a proper communication. Finally, a detailed description of the main interface functions will be shown.

4.1 Entities Created in ARS Model to Provide a Proper Communication

During the course of this work, some new classes were implemented or modified in order to get proper behaviour from the whole system. When an adaptation of ARS system acting as a UT2004 logic bot system was wanted to be done, it is logical to think that new pieces would appear due to data must be introduced and processed in a correct way.

4.1.1 CLSUNREALBODY

In this case, the class clsUnrealBody was created especially for this work. The reason of that is an adaptation of the previous body, clsComplexBody, that was implemented, was not possible and a decision of a new body creation was decided. So, an extension of clsComplexBody was made in order to implement clsUnrealbody. Moreover, the clsComplexBody is an extension of clsBaseBody.

Starting from the last class, clsBaseBody includes all the basics properties or attributes of the embodied agent body. Two main attributes in this class can be identified: eBodyType and clsAttributes. Some kinds of body are available but in the course of this work the body type will be UNREAL. On the other hand, body attributes are defined in the object of clsAttributes that contains a HashMap<eBodyAttributes, clsBaseAttribute> as a main global variable. The eBodyAttributes can be colour, shape, or height.

Going one level up, clsComplexBody that implements clsBaseBody contains the logical part of an embodied agent. It is the basic container for each entity the body needs: internal and external I/O, biosystem and brain. The main class is clsBrainSocket. This class is explained in the next subsection in detail.

The last class in this body architecture is clsUnrealBody that was created for this work. Inside this class two doubles are identified as armor level and ammo level as well. Some methods were created to permit re execution of different users-cases though previously as shows the next figure. To generate the drive nourish, that is responsible to eat in order to regenerate the health of the embodied agent, the stomach of the agent must be empty. When it happens, the brain priories eat action, in our case the embodied agent will "eat" a healthpack. To do that DestroyAllNutritionAndEnergyForModelTesting() must be called. Two methods permit hurt or heal agent body: HurtBody() and HealBody(). Finally, when a healthpack is eaten the last function of the next figure is called.

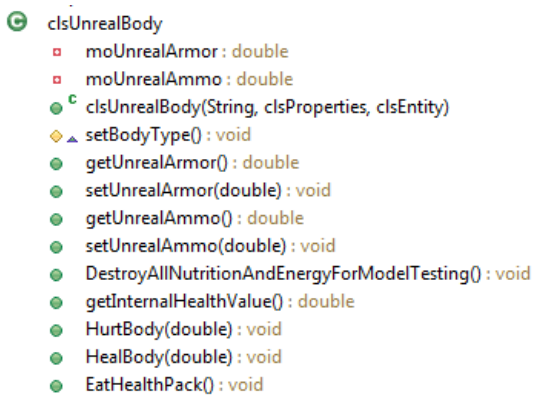


Figure 4.1: Class clsUnrealBody

4.1.2 CLSBRAINSOCKET

One of these classes is clsBrainSocket that will be used by the general interface to get control the logic of the ARS engine in a simple way. As its name indicates, this class will be a socket of the ARS logic system or, in other words, ARS brain. It means that this connection between the ARS and interface will permit, among different things, introduce to ARS engine the perception data collected by the UT2004 embodied agent or convert internal parameters such as stamina or temperature.

Going deeper in this class, some attributes are defined. For instance, four integer constants are defined: UNREAL NEAR DISTANCE, UNREAL MEDIUM DISTANCE, UNREAL FAR DISTANCE and UNREAL AREA OF VIEW RADIANS. All of these constants will permit convert from UT2004 vision system to ARS vision system. There are two more attributes that are important: moSensorsExt and moSensorsInt. They allow Brain socket to store all the information that come from sensors. There is one variable for each kind of sensors: external and internal. Moreover, these two variables are implemented using hashmaps (HashMap <eSensorExtType, clsSensorExt> moSensorsExt). It permits that for each sensor type all information related to it is stored using as a key the sensor type name. The last important attribute is moUnrealVisionValues. This variable is a vector and it contains all the values sended by the interface related to the objects that are being seen by the embodied agent such as healtpacks or enemies.

Talking about methods, some public methods and, as a consequence, some private methods are used by the interface. For example, a first method is processUnrealVision(). This method receives a vec-

tor with all objects that are being seen by the embodied agent an instance of the general variable to store data information, clsSensorData. For each vision ranges (near, medium and far), an internal clsVision object is created in order to store the UT2004 objects. When this is done, a conversion from unreal vision ranges to ARS vision ranges is done. It is necessary remember that unreal vision ranges is based on polar coordinates and ARS vision ranges is based on fields, in one direction is near, medium and far and in the other direction is left, middle_left, center, middle_right, right. To do that an internal method is called: convertUNREALVision2DUVision(). When all this process is done, all information proceed is store in the clsSensorData object.

Until now, only the propertires of the UT2004 objects related to vision parameters are treated. Using convertUNREALVisionEntry(), brain socket establish the other properties: type, if is alive, shape, color and entity ID. From now on, only the properties used to recognize an object will be shape and color. In the next figure a exemplification of how these properties are set is shown.

```
switch(poUNREALSensorVisionData.getType())
{
    case HEALTH:
    {
        oData.setEntityType(du.enums.eEntityType.HEALTH);
        oData.setAlive(false);
        oData.setShapeType( du.enums.eShapeType.SQUARE );
        oData.setColor( new Color(34,139,34) );
        oData.setEntityId(poUNREALSensorVisionData.getID());
        break;
    }
```

4.1.3 CLSUNREALSENSORVALUEVISION

A new class was necessary to be created due to a new kind of information should be stored in the correct way. This information was the UT2004 external vision entries. In other words, the objects that are seen by the embodied agent must contain specific parameters in order to be able to use proper information of them. This class will be shared for both environments: interface and ARS engine.

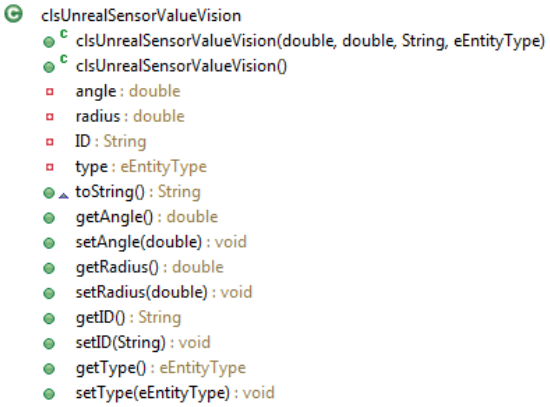


Figure 4.2: Class clsUnrealSensorValueVision

In previous figure, all the parameters and methods of clsUnrealSensorValueVision are shown and, as is possible to see, four parameters will define the properties of these objects: angle, radius, ID and type. The first to parameters will be used by the brain socket to translate the polar coordinates to ARS vision system. The third parameter is used to identify the object and the last one will permit the brain socket set shape and colour of this object.

4.2 Interface

Before define the architecture of the interface, the task of the Interface must be clear and not only it. Another thing must be defined as clear as possible; it is with that information the interface will work and that information will deliver to the other entities.

4.2.1 Interfaces Manager

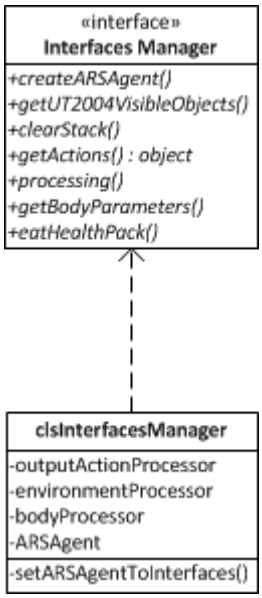


Figure 4.3: UML diagram Interfaces Manager

As it is explained in Interface architecture, in the section 3.4.2, the entity that will be responsible to get or give the proper information from or to ARS model or ARS bot is the interfaces manager. It means that nobody has a direct access to the body, environment or output interface. Thanks to that,

the interface achieves an independency and for the bot and for the ARS model it is a black box. In the previous UML diagram, all the defined functions in the interface class and also in the implementation of this interface are shown.

4.2.2 Body Interface

In the second interface implemented all data related to internal parameters of the agent body are processed and toward to ARS decision unit. It means that body interface is responsible to establish proper information in order to generate the drives. To do that, some methods, that will be explained in this section, are implemented and make possible the connection between UT2004 agent body and the way that ARS concept a body [Mun12].

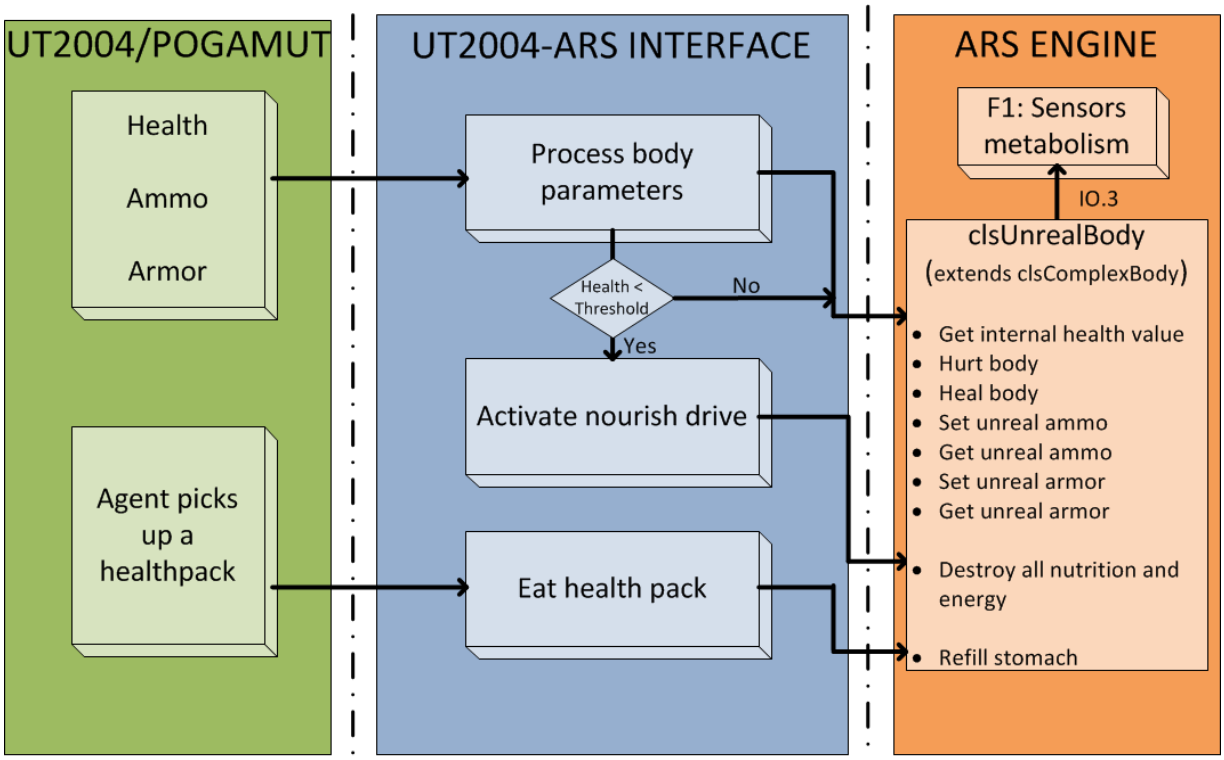


Figure 4.4: Body interface architecture

Knowing that this interface will work in the first step of the whole engine, input part, an essential method takes care to set the basic parameter of UT2004 body: health, armour and ammo. For the user-cases tested in this master thesis, a method used is activateNourish() due to when a the embodied agent is hurt it is not so easy reduce the health in the ARS system. Let imagine this scenario: agent is trying to find an enemy and suddenly it is hurt. When it happens, two main reactions must be produced in the ARS engine: the reduction of embodied agent health and the generation of the

drive nourish. In fact these two facts can be seen as a consequence of each other. To reduce health the interface call an internal method to make empty the stomach of the ARS embodied agent. This situation produced a negative effect on the health and it is going down. This internal disequilibrium generate the need of pick a healthpack up as soon as possible or, in other word, generate the nourish drive. When a healthpack is recollected a method has the task of fulfil the stomach of ARS agent and recover the health.

In the next UML diagram, all the defined functions in the interface class and also the function and attributes in the implementation of this interface are shown.

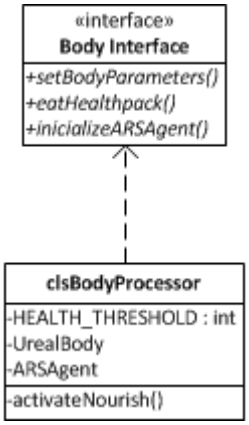


Figure 4.5: UML diagram Body Interface

4.2.3 Environment Interface

The first task of the environment interface is to get from the list of objects, that are being seen for the embodied agent, ready to be interpretative for the ARS engine. To do it, there are three kind of information must be clarified: location and identification. The next figure shows this process.

As a bridge between UT2004 object and ARS the class clsUnrealSensorValueVision have been implemented. A clsUnrealSensorValueVision List is implemented as clsEnvironment attribute. This class is what is known as thing presentation. As it was described in previous chapters, ARS environment contains first process.

The first property of an object must the location. In order to get angle and radius parameters a simple algorithm had been implemented. The point is that the location of the UT2004 object inside a map must be relative. When the method getLocation() is called, a Location entity is received. Each Item has a fixed Location in order to be situated [Mun12]. For this reason, embodied agent location is used to get the relative position of the item. So in that moment angle and radius are assigned.

When this process is done, two more properties must be assigned: ID and type. Every possible object type is defined in a configuration file that permits a registration of all objects that are in the embodied agent environment. The ID property it is use to introduce another classification. This process is done for the items and enemies as well.

The case of how walls are detected or perceived is different. As a part of this work, it will be explained in detail and trying to give the general idea due to it was implemented by the ARS model but it is not used yet. However, it will be a good start step to be able to implement in the future an efficient navigation system. It will be necessary in a future to introduce this functionality inside the ARS engine because until now it is not done.

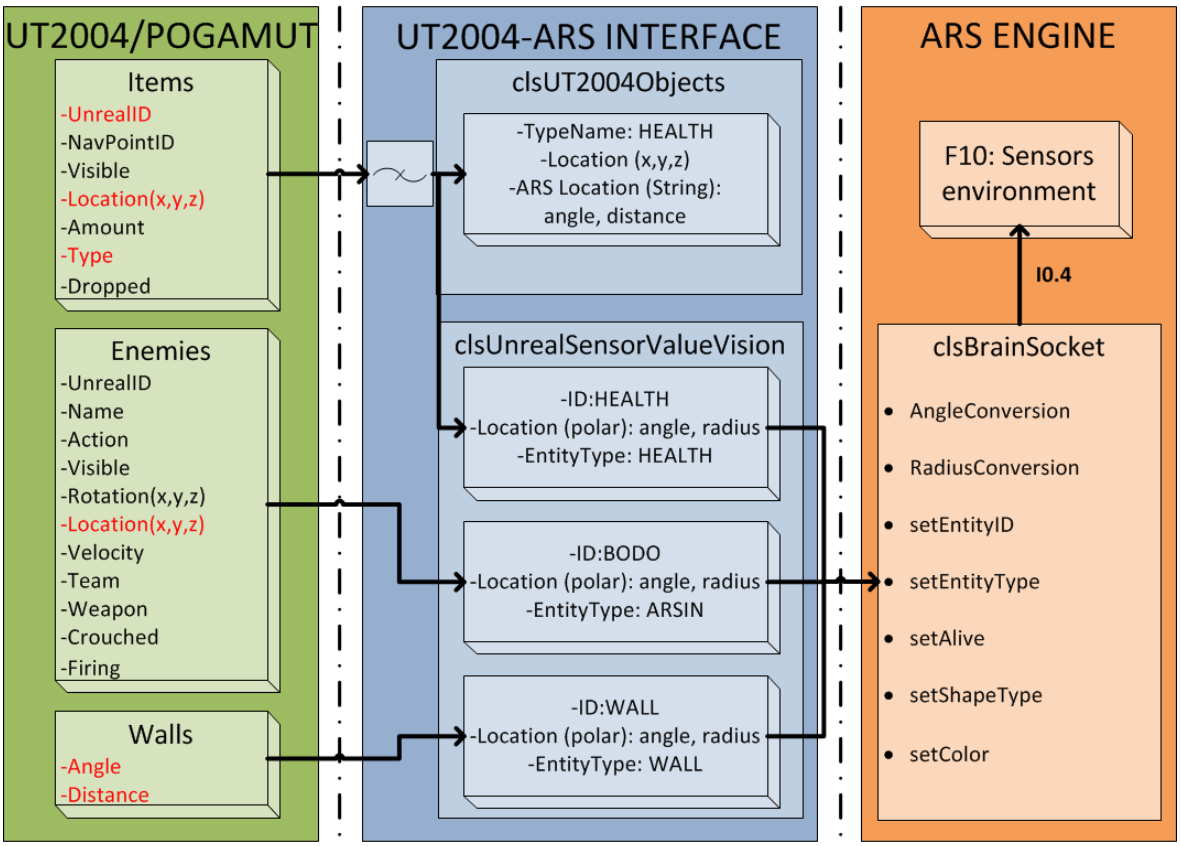


Figure 4.6: Environment interface architecture

To detect them, ray casting sensors are used. A ray casting is a line that plays the role as the vision and gives information about something is in its direction or not. Basically, this something should be a wall entity or something that will be defined as a wall due to it is just an obstacle for the agent. The point is that ARS vision is how it is implemented, so walls must be settled in one of these fields.

As reader remembers, if not check the section 2.2.1, the ARS vision follows two features to split the visible space: radius and phi (polar). Due to as all field of ARS vision must be covered, for each

polar section there is a big ray casting that in fact represents three (each one for the radius sections). So the embodied agent is using 15 rays casting to detect the walls.

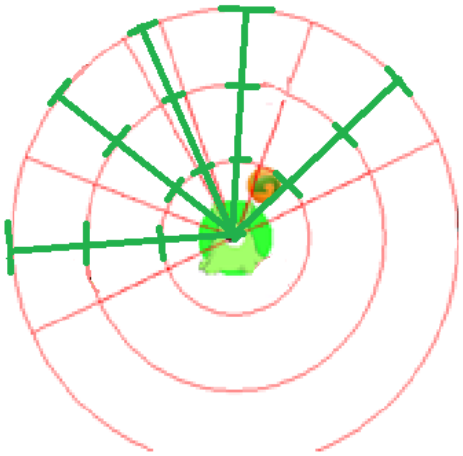


Figure 4.7: Ray casting sensors distribution in ARS old vision system

The process is this: the agent is moving in the map randomly and when a wall is next to it some sensors will be activate. The first sensors that are checked are the longest ones (radius section far), if there is one activated, shorter ones will be checked as well to get the near position of this wall. It will be located in the middle of the field due to the position of it is not exact.

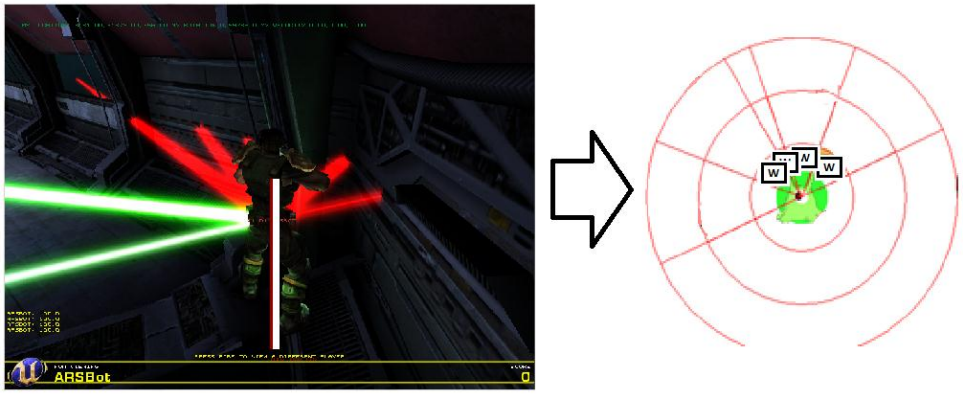


Figure 4.8: Matching both vision systems

In the next UML diagram, all the defined functions in the interface class and also the function and attributes in the implementation of this interface are shown.

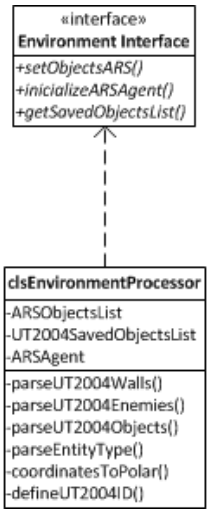


Figure 4.9: UML diagram Environment Interface

4.2.4 Output Interface

Another goal of this master thesis was the implementation of an outputs interface that would be responsible to get the action from ARS engine and convert them to UT204 actions. All this process could seem a kind of trivial work but it is not. In this section, this process and the logics followed will be explained.

The point of the outputs interface is when the decision making selects the action to be done by the embodied agent; the interface takes them from brain socket. There the actions are in a XML format, first the action name is introduced and then possible parameters of this actions are introduced as well (for instance; degrees to turn or speed to move). This XML string is parsed and name and parameters' action are used to create a clsUT2004Action entity. This class contains different attributes (it not compulsory used all of them, just parameters needed for each action). In the case of the action UNREAL_MOVE_TO the location is needed so it is taken from UT2004SavedObjectsList to retrieve the exact location of the items compering with the ARS location (give it in XML string).

Until now, only there are three actions that can be received:

- Search
- Unreal_move_to
- Flee

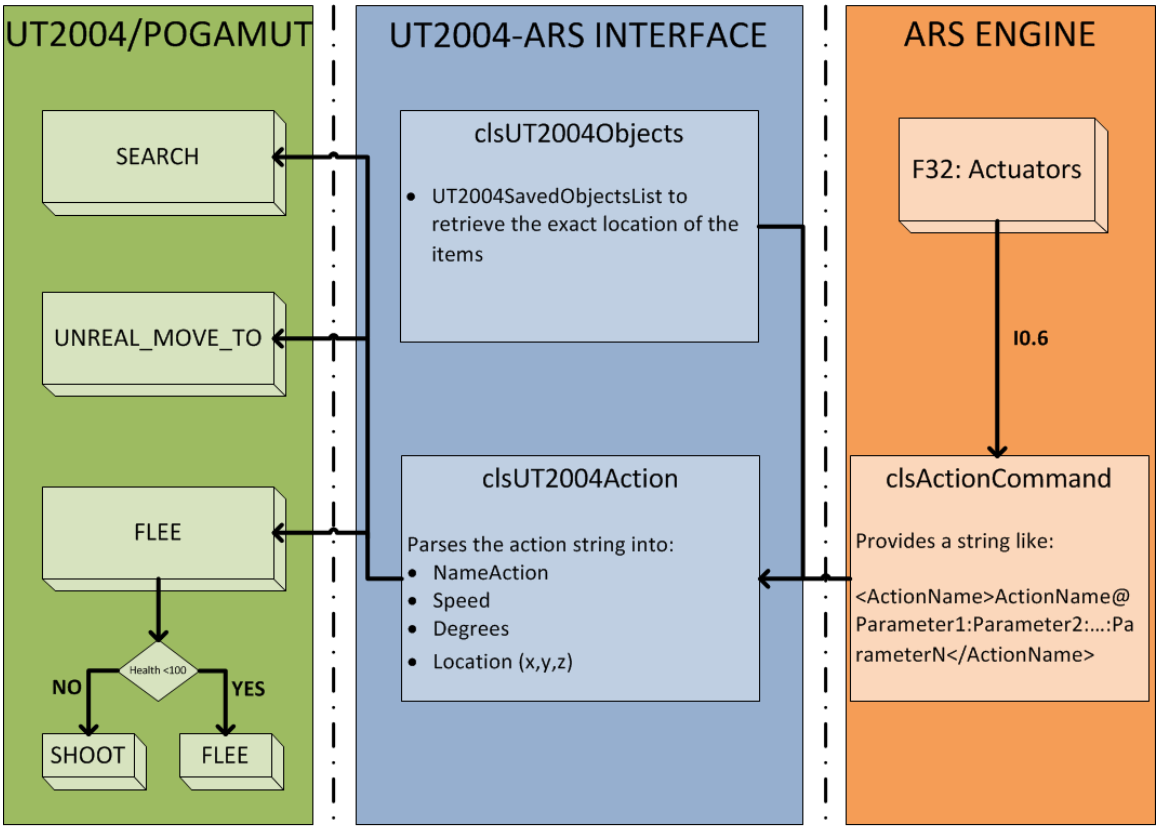


Figure 4.10: Outputs interface architecture

The first possible action is search. This action is designed to be in fact a command sequence that can contain simple actions such as move forward or turn in some direction. As it is explained in the section 4.1, this command sequence is not develop enough due to it always gives actions move forward and turn right. For this reason an algorithm is implemented in Pogamut side based on ray castings sensors to select the good path.

The second possible action is Unreal move to. It is related to nourish drive and give the functionality to go to an exact point of the map and pick up whatever is there. Until now, only health packs are picked up by the agent. To get the exact location of an item, the interface check the exact position of it using as match condition the ARS location from the XML string.

The last action is Flee. It consists in two actions (due to shoot action is not implemented yet in ARS model): flee and shoot. When the flee action is received in Pogamut side, the health level is check and depending of this condition the agent will turn back and go away or it will shoot the enemy. This action is related to panic emotion.

In the next UML diagram, all the defined functions in the interface class and also the function and attributes in the implementation of this interface are shown.

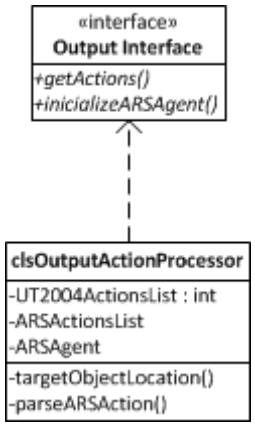


Figure 4.11: UML diagram Output Interface

4.3 Main Interface Functions

A detailed explanation of the main functions of the Interface will be explained through UML diagram to show how they work in this section. There are three main functions, two of them are to set inputs in ARS model and the third one is to get the outputs.

4.3.1 Set Body Perceptions

When from the UT2004 side the body perceptions are wanted to introduce to ARS agent, the only way to do it is using the method getBodyParameter() from clsInterfacesManager. The health level and the ammo level are sended as parameters and these information is sended to clsBodyInterface. There, the information is processed and introduced in ARS agent body.

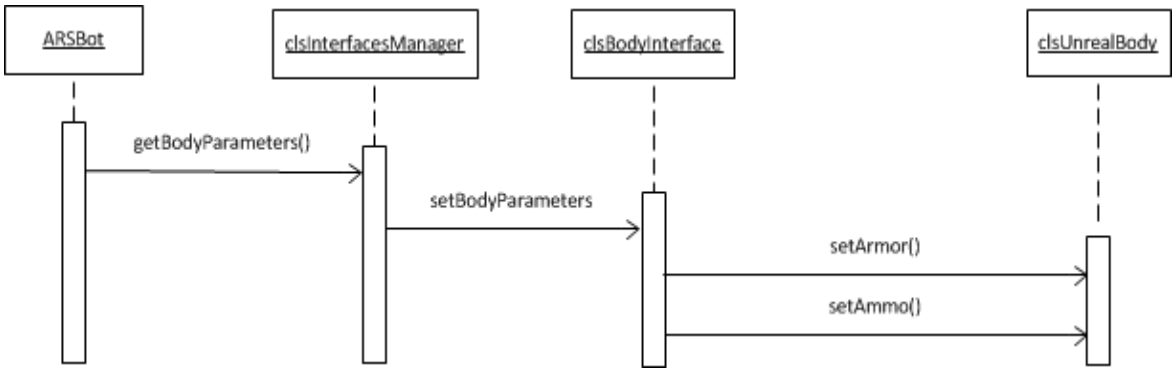


Figure 4.12: UML diagram set body perceptions

4.3.2 Set Environment Perceptions

To do the process for the environment perceptions, the ARS bot will communicate this intention to the clsInterfacesManager and, in this case, the information will be send to the clsEnvironmentInterface, that is the entity responsible to prepare this information to be ina proper condition to introduce them to the ARS agent body.

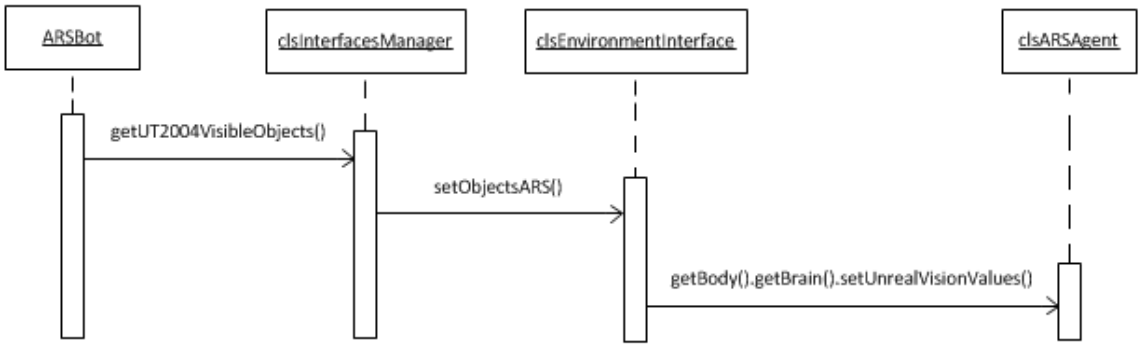


Figure 4.13: UML diagram set environment perceptions

4.3.3 Get Actions

In this last function, the goal is to extract the actions that the ARS decision making decides. When the ARS bot shows this intention usinf the method getActions() from the clsInterfacesManager, it will call the method with the same name form clsOutputInterface. Once the actions are received form the ARS agent, the clsOutputInterface converts these ARSActionList into UT2004ActionsList in order to prepare them for the ARSbot, who receives it from the clsInterfacesManager.

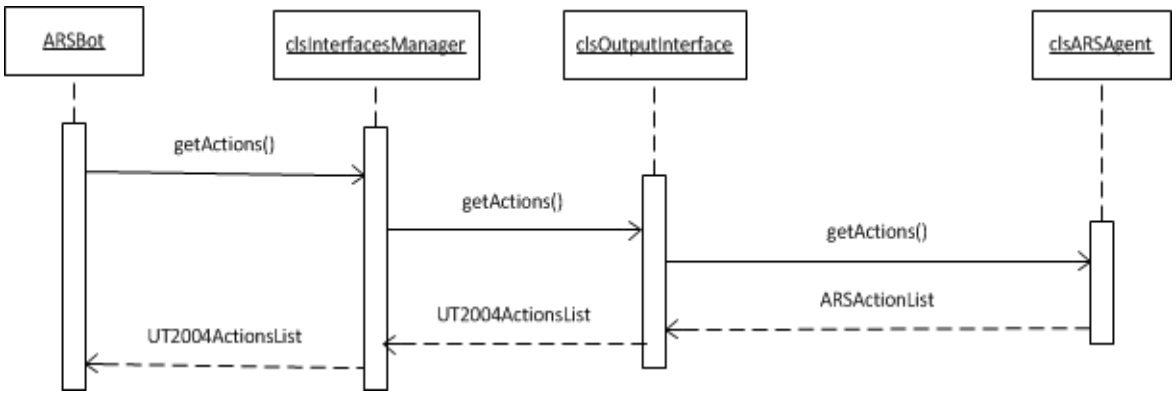


Figure 4.14: UML diagram get actions

5. Use-cases

Obviously, all functionalities of the Interface are tested using use-cases. Each use-case is designed to test a particular functionality. A detailed explanation will be given including the main terms used and a technical description, which will help to understand the process of each use-case.

5.1 Navigating on the map

In this first use-case, a feature of the agent is wanted to check: the ability of the agent to navigate within the current map in a random way in order to not be stopped. It gives to the agent autonomy due to this navigation algorithm is based on the senses. It means that the navigation points inside the map are not used by the agent to go from one place to another one.

Unfortunately, this functionality is not implemented in the ARS side due to the lack of resources that it gives. It is for these reason that the navigation algorithm is implemented in Pogamut side, expecting that it will give an idea for futures works to basing on this code be able to implement one module that will give the same functionality as this one.

5.1.1 Main Terms Used

Search High level Action

In this first use-case no new action is implemented to activate the search functionality. The action search from the first ARS simulation is reused due to it gives the resources necessaries to the Pogamut side. In fact, this search action is a command sequence, it means that it is not a simple action but is a composition of simple action that all together represents the high level action search.

Simple actions that compose the search action are: move forward, turn right and turn left. Each of them provides the associated parameter in XML string action, for instance in the case of move forward the associated parameter is agent speed or how many degrees must turn in the case of turn. Although the fact to not be a new gives facilities due to everything is done, it has provides a bad behavior of the agent.

The point of this command action is that it always provides the same simple action that were enough for the 2D world from first simulation for the ARS but in the UT2004 environment, that has all the complex specifications of a 3D world, an improvement of this search is required in order to get a proper navigation appearance from the embodied agent in the map.

Ray castings

Due to simple actions received inside the command sequence are not sufficient as it was explained before, a new algorithm was created to take care of the agent navigation. The main entity of it is the ray castings that will allows to embodied agent possibility to move in a humanlike way due to they will act as sensors to detect walls and give the proper information to avoid to crash into them.

From all the ray castings, the navigation algorithm only uses these ones that have the length equal to the threshold near. So in total six sensors are used to extract proper information from the agent situation and that action (move forward, turn left and turn right) should be done in order to have as realistic as possible navigation.

5.1.2 Technical Description

There are two main points in this use-case: ray castings and agent navigation algorithm. Both have the same importance in the technical description due to both task are indispensable in the realization of this use-case. It is important to remark that this functionality shown here is in the UT2004 bot side so in future works it must be implemented in ARS model in some way.

Firstly, inputs will be generated using the ray castings as body sensors. They can be seen as eyes of the agent due to they will be responsible to give the proper information to the algorithm. This information is formed by a set of booleans. Each boolean represents a ray and when the value of it is true it means that a wall is in the same direction of the ray.

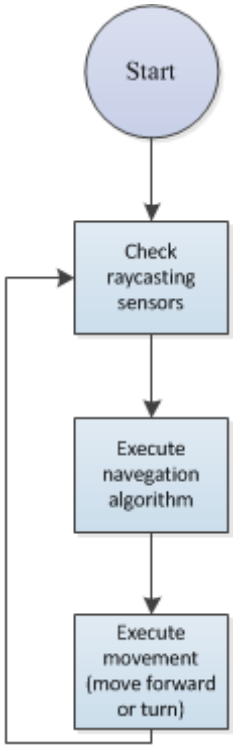


Figure 5.1: Flowchart Navegating on the map

Secondly, the navigation algorithm must decide what the agent must to go using the information received from the sensors in order to not crash with the wall. The first sensor value checked is the sensor front, that is a combination of the two centered ray castings values. If nothing is detected, the action decided to do is move forward, if a wall is detected then the other sensors are checked and depending on the value it will turn left or right and it will turn 15º or 30º. In the next figure a diagram of the algorithm is shown.

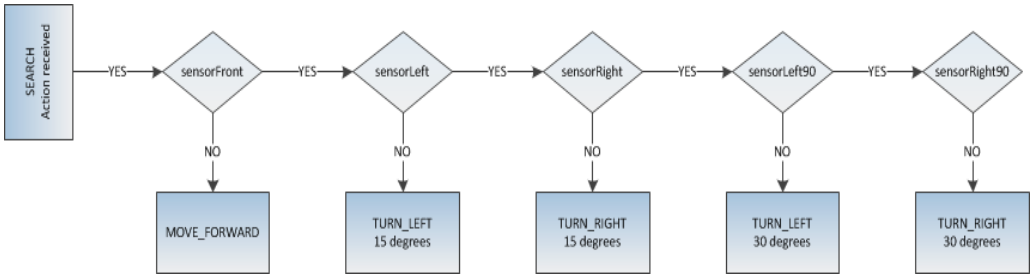


Figure 5.2: Navigation algorithm

5.1.3 Results

When the navigation algorithm, as well as ray castings, is tested in the UT2004 environment a good performance of them can be seen. In fact, the decision to set the small turn in 15º and the big one in 30º give an excellence outcome due to allows the agent to overcome walls that appear in front of the agent. These values were selected based on the prove-error method. Even that the interface introduce inside the ARS engine walls that are being seen by the agent, the model is not develop enough to give a proper direction commands. Although the algorithm using ray castings where developed, it is still not sufficient. For that, even if the ARSDU decides that it is time to satisfy the nourish demand, can be possible that the agent will not receive the action move to due to it will not be able to see a healthpack.

5.2 Generation of Nourish Drive and Pick Healthpack up.

The purpose of this second use-case is to test the agent ability to realize her current health value in a specific situation and heal her body when it will be necessary. To do that, the generation of the nourish drive will be essential in order to get the action that will permit receive the proper action to pick up a healthpack from ARS decision unit.

There are two covered aspects in this use-case. Firstly, the possibility to heart the agent and, when it happens, prepare the proper data to introduce them inside the ARS engine. Secondly, when the decision unit decides that is time to pick up a health pack, the agent action that permit do it is proved.

Obviously, there are some aspect that are not covered in this use-case in the sense of may be where taking in consideration when it was designed but unfortunately were not implemented due to lack of resources or time. One of them is that before all these process of generation of the nourish demand and the collection of the health pack a delay is introduced by ARS decision unit to give the action unreal_move_to and, as a consequence, the agent execution is not so fluent.

5.2.1 Main Terms Used

Homeostatic Parameters

As it was explained in different sections of this document, homeostatic parameters play a main role when ARS is prioritizing all drives, setting drive demands, in order to be able to take a decision. In particular, some of the homeostatic parameters that will be evaluated and, for this reason, that must be modified in the correct way are: carbohydrate, fat, mineral, protein, vitamin and water.

```
public enum eNutritions {
    PROTEIN,
    FAT,
    VITAMIN,
    CARBOHYDRATE,
    WATER,
    MINERAL,
    TRACEELEMENT,
    VITAMIN_A,
    VITAMIN_B,
    VITAMIN_C,
    UNDIGESTABLE,
}
```

Figure 5.3: enum eNutrition

The class clsUnrealBody allows control features in the instance of unreal agent inside the ARS architecture. To do these modifications or actualizations of the body agent some methods are implemented as is explained in the section 6.3.2. One of them is called DestroyAllNutritionAndEnergy(). In the next figure the architecture of this method is detailed but basically what it does is get the internal system from the complex body and then from there it gets the stomach system. The stomach system has a method called DestroyAllEnergy() that take all the nutrients define in the enum eNutrition and, one by one, try to reduce their quantity and if it is not possible redefine them giving the initial value that is zero.

As was explained, when the method DestroyAllNutritionAndEnergy() is called the stomach of the embodied agent is emptying, and because of that nourish drive is activated by the id due to it feels the need to get energy to be able to move, the method eatHealthpack() that is in the class clsUnrealBody is called in order to reestablish normal values in the homeostatic parameters. When the values are reestablished, the id part of the mental process assign again a low priority in the nourish drive.

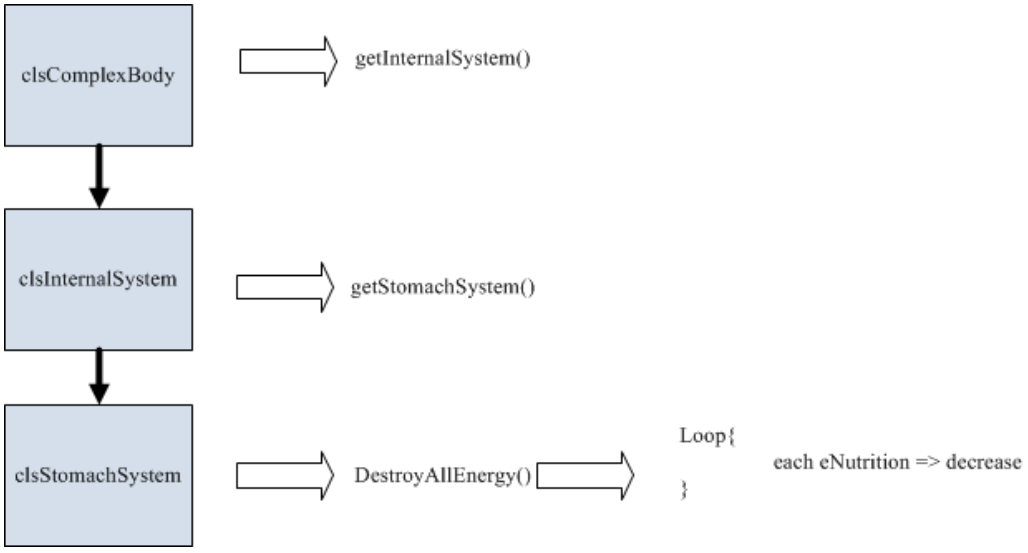


Figure 5.4: Heretical of clsUT004body

Healthpack Object

When an object is used by ARS decision unit, it must be defined previously with the proper parameters. In this use-case only one object is required to be introduced on the system in order to give to ARS engine the possibility of assigned it as an object of drive, as it explained in the section 5.3.2. This object is a healthpack and it will allow the embodied agent to satisfice the nourish drive.

As was also explained in detail in previous sections, an object in ARS environment is defined using basically two properties that will be used by ARS decision unit to recognize them in the mental process: shape and color. In this case the shape selected was a square and the color in RGB code was [34, 139, 34]. Another parameter used is a boolean that is used to set if the object is alive or not so, in the case of the healthpack, is false.

clsActionUnrealMoveTo

When this use-case was raised a dilemma appeared: was it necessary to create a new action? The answer was affirmative due to special parameters were required by the interface in order to do the embodied agent movement: entity and position. Both parameters will be needed to know information related to the object that will be picked up: first, to know what object is required by the agent and, second, where is this object. In this first use-case, the parameter entity will be a healthpack but this action will permit the implementation of more use-cases for instance, a possible future scenario will be when the agent will need to pick up weapons to be ready for shooting.

All the action defined in ARS decision unit, and the class clsActionUnrealMoveTo is not an exception, implements the abstract class clsActionCommand. This abstract class contains the method getCommands() that permits get an array list of commands, clsActionCommand. It is used when the

action is a high level action. For example, when the action search is received, the interface must get the simple actions (move or turn) that all together build the high level action. In the case of clsActionUnrealMoveTo this method is not used. Also if it is nessesary a method check if the action is completed using isCompleted() and the identification string can be got it using the method toString(). This string is received in XML format.

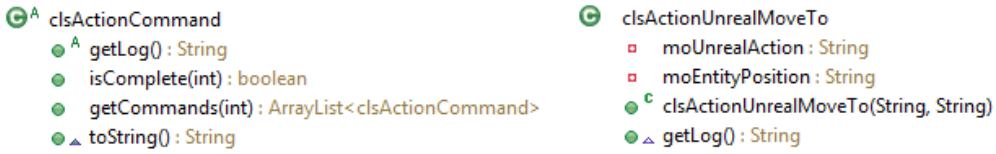


Figure 5.5: Classes clsActionComand and clsActionUnrealMoveTo

If the class clsActionUnrealMoveTo is analyzed two parameters that are strings can be identified: moUnrealAction and moEntityPosition. Both permit get the information defined before, and together with the abstract class of the clsActionCommand give the tools to process it by the interface. In the next figure the identification string in XML form received from the ARS decision unit is shown.

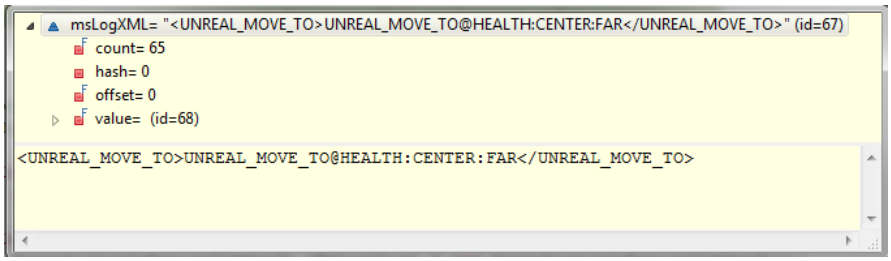


Figure 5.6: XML action string

5.2.2 Technical Description

Inputs

HEALTH-PACK OBJECTS

Two main inputs must be set in order to provide the required information to be able to do this first use-case: healthpacks that are seen by the agent and the health level of it. Both will be collect firstly in the logic method of the pogamut bot. To get objects that are seen by the agent the command used is: bot.getWorldView().getAllVisible(Item.class).values(). On the other hand, the command that is called to get the health level is: info.getHealth().

When the collection of items are got, it is send to the interface manager using the method getUT2004VisibleObjects() together with the location of the agent due to the pogamut bot will only have a communication with the interface manager in order to isolate the pogamut and the interface created in this work. The interface manager will send these two parameters to the environment interface through the method setObjectsARS(). On it the object will be treated, thanks to the private method parseUT2004Objects(), due to the parameters received from pogamut must be adapted to ARS parameters.

Basically, a main entity will be created in this private method: a vector of the class called clsUnrealSensorValueVision. This class contains the information required by the ARS decision unit for the definition of ARS objects: ID, type and position in polar coordinates (the pogamut environment provides objects in cartesian coordinates). When this staff is done the environment interface will send this information to the brain socket through the variable clsARSIN that instantiates all the ARS decision unit. All these objects are stored in local virable called UT2004SavedObjectsList that is a Vector<clsUT2004Objects>. This variable contains the name of the object, the unreal location and two strings with permit defined the ARS location, for instance “CENTER” and “FAR”. This variable will be used to recognized that abject the ARS decision unit have been selected to satisfy the current demand, in this case a healthpack and the nourish drive. In the subsection Outputs more details will be given.

HEALTH

On the other hand, the health level will be send to the interface manager using the method getBodyParameters(). In it, the interface the method setBodyParameters()manager is called in order to set this parameter where through the class clsUnrealBody created specialy for this work will set the internal health value and if it is below a threshold, the method activateNourish() will be called in order to receive the action to pick up a healthpack.

How these inputs travel inside the ARS engine

HEALTH-PACK OBJECTS

Once the UT2004 objects are introduced in the ARS engine using the proper method in the brain socket, they must be processed in order to adapt them into the ARS environment and use them properly. This data processing is done in the modules responsible to treat environmental perceptions: F10 (Sensors environment) and F11 (Neurosymbolization environment). In F10, the inputs objects are stored in a attribute called moEnviromentalData that is a HashMap<eSensorExtType,clsSensorExtern>. As a key of the hashMap, an exterior sensor is used to store the objects that contains all the information required (shape, color and position), for instance vision_far. This attribute is send to the next module, F11, that holds the sensortype and the sensor symbol (converted from the extSensor value) and stored in a HashMap<eSymbolExtType, itfSymbol>.

When this is done, it is time to introduce the perceptions received from the inputs interfaces into the memory. In F14 (External perception), neurosymbolic contents are transformed into thing presenta-

tions. This module has an attribute an ArrayList <clsPrimaryDataStructure> to store all TP generated where if the agent is seeing a health-pack one of these TP will represent it. This information is send to F46 (Memory traces for perception) where an association of TPMs (TP with emotion and fantasies) with thing presentations raw data (from external perception), so, in other words, a healthpack object is associated with the idea of a healthpack.

In that moment the thing presentations that contain health-packs are ready to travel through the rest of primary process modules until the module F21 (Conversion to secondary process for perception) where the correspondents word presentations will be created. One by one the classes' clsThingPresentationMesh are converted to clsWordPresentationMesh where in the case of the health-pack the word presentation contains an association with the health-pack object that has the parameters required to be identified as a demand object of the nourish drive.

HEALTH

In the case of health level, when it is set in the clsHealthSystem object in the class clsUnrealBody, two modules are responsible to take care of this information that are: F1 (Sensors metabolism) and F2 (Neurosymbolization of needs). As is explained in the section 5.3.3, these modules prepare the homeostasis parameters that will be used to build the self-preservation drives.

The sensors of module F1 collect information on bodily functions like health or blood pressure. Thus the current state of the body and its need is made available. However, no real processing is made in this module as the information is actually collected. F1 contains an attribute to do that that is a HashMap < eSensorIntType, clsDataBase> known as moHomeostasis. Inside the enum eSensorIntType there is declared the type health, so is there were the health level is stored as a parameter. In fact, the information is stored inside the class clsHealthsystem that implements the class clsSensorIntern and this class is an implementation of the class clsDataBase.

Once the task of F1 is completed, the information is send to F2 where a conversion of homoestatic data into neuro-symbols is done. The main attribute of F2 is a HashMap<String, Double> called moHomeostaticSymbol. Here, the key of the hashmap is a string, that in this case is HEALTH, and the value is a double that represents the tension of a specific symbol. An interesting point is that the value of the health is stored as a percentage (to maintain independency with inputs).

Activation nourish drive

The hot point in this use-case is to achieve the activation of the nourish drive when the situation will demanded it. Of course, to possible situations must be studied in this subsection: first, when the health lever is upper than the threshold imposed and second, when the health level is lower than it. The reactions of the body demands, drives, must be completely different each one.

The generation of some drives, included the nourish drive, is done in the F3 (Generation of self-preservation drives) and F4 (Fusion of self-preservation drives) and later, treated all this information

in the F48 (Accumulation of affects for drives) and F57 (Memory traces for drives). Once all this modules are remembered from the section 5.3.3, let see the two possible scenarios.

There are different points inside the model where the dataflow related to drives cab be checked. For instance, inside the module F3 in the attribute ArrayList <clsPair <clsDriveMesh, clsDriveDemand> called moDrives where drive demand can analyze. To compute this data, the nourish drive demand, all the homeostasis parameters such as blood sugar. The ARS engine is designed following a human-like behavior as it is clear in this advanced part of the document, so it means that the blood sugar lever is going down due to the is the normal behavior in a life being. In the case of unreal tournament in every loop of the ARS engine this parameter must be incremented with the same level that is decremented in order to don't get an unexpected conduct of the model. It means that the bot only expect receive the action move to in one situation: when the health level is under the threshold.

On one hand, when the health level is upper than the threshold, the drive demand must not be selected as the main drive to be satisfied. So, as it is explained before, the drive demand is increased as in a slowly way. On the other hand, when the health level is under the threshold, the interface takes care to make empty the stomach of the agent and it makes the blood sugar decreases fast and, as a consequence, the nourish drive demand is increased in the same way.

Output

Basically, if the generation of the outputs is wanted to be debugged, the module F26 (Decision making) must be checked. There, Demands provided by reality, drives, and Superego are merged. The result is evaluated regarding that resulting wish can be used as motive for an action tendency. The list of produced motives is ordered according to their satisfaction level that is able to give to the metabolism. These motivation or goals are stored in an ArrayList<clsWordPresentationMesh> called moGoalList_OUT. So depending if the nourish goal is selected two possible actions can be send through F31 (Neuro-desymbolization of action) and F32 (Actuators) to the interface and, from there, the UT2004 bot.

So in normal condition, that are the bot is not heart, the action that will be decided to be executed by the embodied agent as the most adequate for that moment will be the action search . This action gives the idea for the interface to move randomly in the map. On the other hand if the bot have been hurt and, as a consequence, the nourish drive have been activate due to the nourish drive have a high level, the action selected to be executed is the action unreal move to, where the content health and the position of it will be included. All these actions are saved in an ArrayList < clsActionCommand > known as moActionCommandList_Output.

Once it is done, the outputs interface get the actions from the instance of arsin that have a method to get the proper action from the command stack from the action processor and before from the external inputs and outputs. When the action are received from ARS decicion unit, they are procced by the outputs interface in order to get the information properly due to in the ARS way it is impossible for Pogamut to understand it. In this sense, a class called clsUT2004Action gives the possibility to

do this work. As an attributes has all possible parameters related to actions that can be received from ARS side and depending that action is some parameters will be used.

When the health is lower than the threshold, the action received for the interface is MOVE_TO. In this other case, the interface is responsible to extract the information properly: what the agent must pick up and where it is. As is explained before, the ARS vision resolution is so low and only gives the field where, in this case, the healthpack is, for instance left:far. For that reason, the interface must save the objects in order to be able to recuperate the exact location of each object. To do that, the class clsUT2004Objects was created and some attributes were declared: typeName (String), location (Location), radius (String) and phi (String). Thanks to that, when the decion making decides to go for a healthpack that is in the left:far, the interface check all the healthpacks saved and give the action to the agent to go to the location and pick up the healthpack that the attributes radius and phi match.

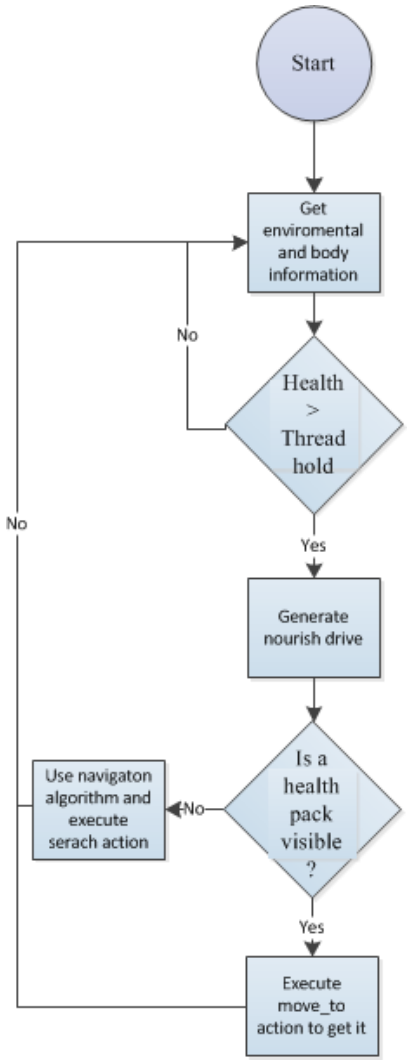


Figure 5.7: Flowchart generation of nourish drive and pick healthpack up

5.2.3 Results

The main goal of this use-case was to generate the nourish drive and it is achieved thanks to two entities: the interface manager, the brain socket and the unreal body. The interface manager, as well as the body and the environment interfaces, create and maintain a successful communication between the UT2004 and the ARS environments. The other two entities, brain socket and unreal body, work properly giving the functionalities for what they were created. In the first case, the brain socket gives the possibilities to introduce the objects perceived by the agent properly and, not only this, make the conversion between the UT2004 vision and the ARS vision. In the second case, the unreal body accomplishes the goal of makes empty the stomach in order to decrease the blood sugar.

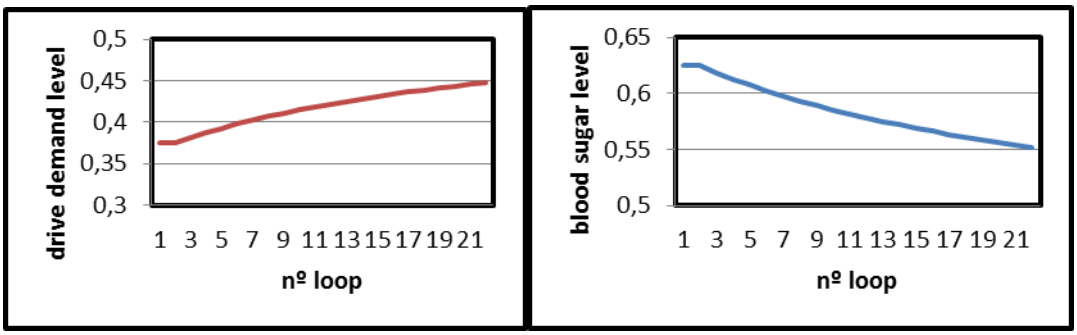


Figure 5.8: Drive demand and blood sugar graphs

In these two first graphs, some feature of ARS engine wanted to be shown and remarked due to are important when someone tries to understand it and get an idea. First, when ARS engine is launched, the stomach is not full and for this reason the blood sugar level is not the maximum value, that is one. So if the blood sugar level is not the maximum level, the drive demand cannot be the minimum value. Another thing is that the homeostatic parameters are decreasing and, as a consequence, the blood sugar level as well in every loop in the ARS engine. This feature implies that the drive demand level is increasing and there is one moment that, even if the agent have not been hurt, the nourish drive is selected to be fulfilled and a health-pack is picked up.

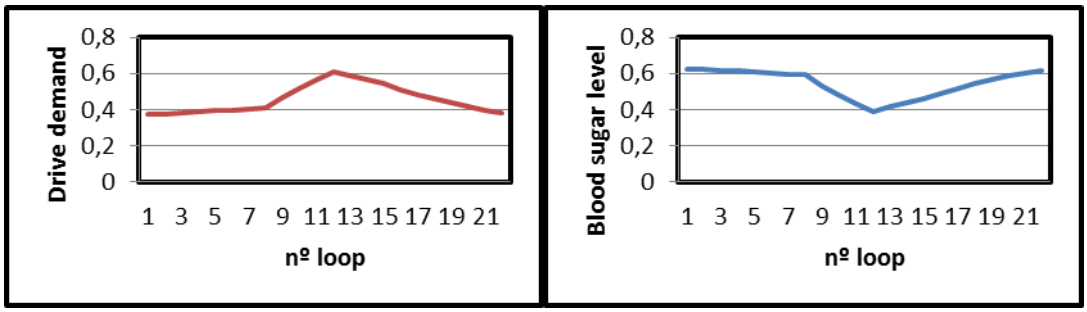


Figure 5.9: Drive demand and blood sugar graphs

More related to this use-case, these graph show how the blood sugar and drive demand react in front of two situations: first when the agent is hurt and second when the agent pick a healthpack up. In this example, in the loop nº 8 the agent is hurt and in that moment the stomach is beginning to empty. When it happens, the blood sugar level is starting to decrease much faster. It implies that the nourish drive demand is increasing much faster as well. Until the loop nº 13 when the agent picks up a healthpack and the stomach is filled. After that, the parameters of the blood sugar and drive demand start to go to normal values. However, this transition is not immediate and takes some loops.

5.3 Generation of Panic Emotion and Shooting or Fleeing

The purpose of this last use-case is to prove two essential abilities of an embodied agent who is in a war environment: the ability to kill, that is the same to say the ability to shoot, and the ability to flee from an enemy when the situation can lead to death of the agent itself. These two agent abilities will permit it to be able to be in the map with other opponents.

5.3.1 Main Terms Used

Enemy Object

Although when the agent see an enemy must now that it will be a possible target to be killed, it will be introduced in ARS engine as an object. In fact, there was one object created for the first ARS simulation that was called Bodo. The entity Bodo was implemented to represent the other unfriendly agents that can be in the map sharing space-time with the embodied agent.

As every ARS object, the Bodo object must to have some properties defined. The entity type of this object is an Arsin. The two parameters that define an object are shape and RGB color and in this case they are a circle and [204, 0, 0]. The main diferent with the other ARS objects is that in this case the boolean that if the object is alive or not is set to true.

Flee Action

When the ARS decision unit decides that it is time to flee, the flee action is selected and, as a consequence, send to the interface. It is a simple action and no parameters are attached in the XML string. As the action search, it does not provide where the agent must flee so the random navigation algorithm is who says where the agent must go. Moreover, the shoot action is also not implemented in the ARS model so when the flee action will be received, in the Pogamut side will be decided that of these two action will be selected to be done by the agent.

5.3.2 Technical Description

Unlike other actions presented in this work such as move to action that is related to a drive, particularly to the nourish drive, flee action is related to an emotion from what is perceived by the embodied agent and not with drive. Specifically, the emotion, that is generated when the situation requires the flee action, is panic. Panic represents the mind need to run away from the current point of the agent due to; in this case, the agent life is in danger. For this reason it must have priority in front the demands of the sexual or self-preservation drives.

The point of this use-case follows this idea: when the agent is navigating randomly in the map and suddenly it sees an enemy the panic emotion will be generated and, because of that, the decision making will decide that the proper action to do is flee due to it is the action with more priority. So, in the interface the flee action will be received and then, due to the shoot action is not implemented in the ARS model yet, the interface will check the health level, if it is over a threshold the agent will shoot the enemy, if not it will flee.

The flee action is based on two actions: first, the agent will make a turn of 180º in order to be in the opposite direction of the enemy and, second, during some steeps it will follow the random navigation algorithm used on the search action in order to go away from the point where the enemy was seen for the last time. Because of this use-case the decision making suffer some changes that are represented in the next UML diagram in order to give the proper functionality.

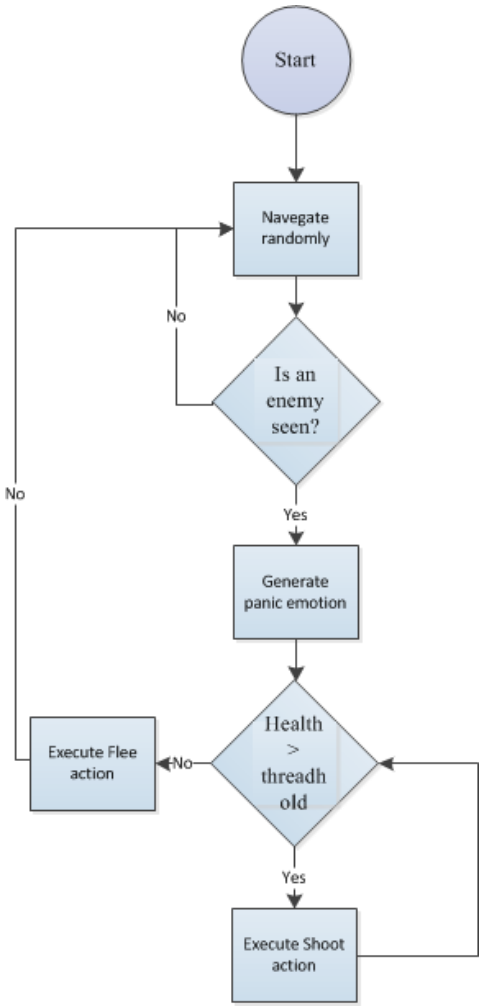


Figure 5.10: Flowchart generation of panic emotion and shoting or fleeing

5.3.3 Results

The main goal of this use-case was generating the panic emotion and, as a consequence, the flee action. A secondary goal was to give to the embodied agent shooting functionality. Both are completely achieved and are tested successfully in this use-case. It shows the proper behavior required in a war environment where you must be able to attract if you think that you have options to kill the enemy or, on the contrary, to flee if you think that the enemy will kill you.

6. Conclusions

At last but not the least, the conclusions of the master thesis will be discussed and also a future work will be proposed answering the question: *Where do we go from here?*

6.1 Discussion

To sum up this work analysed and explained in this entire document, it must be said that the main goal of this project is achieved: to guarantee a successful connection between two environments: UT2004 and ARS decision unit. As it is explained in the section 6.4, where the interface is detailed, the successful connection is shown due to the inputs are introduced in a correct way and the outputs are extracted also correctly.

Another goal was to achieve, not as much prior as the previous one but important as well, it was to achieve a connection as independent as possible. It was done by the interface due to not classes from the Pogamut side were introduced inside the ARS model and neither on the contrary. In other words, a clean connection is settled in order to provide a good communication between the two environments.

At last but not the least and as a consequence of this work, even it was not an explicit goal, before the interface was created an improvement of the ARS model was notice due to several stuff was done. For instance, the creation of new action will permit ARS be more complete or the last use case give the possibility to redesign the ARS decision making giving a step forward. Also the opportunity to test the model in other environment more complex makes it better inevitably.

6.2 Future Work

Once this work is finish and a general vision can be done in the ARS model in order to be able to improve it day by day, some aspects will be treated and commented in this section. Obviously, not all the points explained will have the same impact in the model, so, because of that, this section will start with them that, according to the writer, could cause more repercussion.

6.2.1 Improve ARS Vision

As was indicated in some points on the course of this work, a first weakness that can be improved is the system used by the ARS to position the object that are being seeing by the embodied agent. It was a vision system useful for the first simulation in a 2D world in the ARSIN environment and it was giving the functionality required by this environment. However, when it is wanted to be used in a 3D complex world, unfortunately it is not sufficiently developed because of the resolution.

The point is of this problem is that the ARS vision has a low resolution, so when the environment has a reduced dimensions the vision based on grids is valuable in the sense of functionality. Nevertheless, when the same vision system is applied in a new world where the magnitudes are much considerable, the resolution needed is much more precise in order to known the exact position of the objects and other agent, otherwise the agent is only having an idea of where is the item on the map but not the accurate position of them.

So provisionally, this functionality has been absorbed by the interface as is explained in the section 7.1. Basically, the environment interface keep in an attribute the objects and the enemies that have been seen by the agent in each moment, and then, with the information received from the ARS decision unit, a match between the objects saved and the object selected is done. In a future, this functionality must be implemented inside the ARS engine.

6.2.2 Improve Navigation

Related to the first point of this section, another once comes in almost parallel way: the improvement of the agent navigation. Again, the step forward of passing from a more or less simple 2D environment to a complex 3D world, shows the weakness of the model in the navigation aptitudes and the need to create functionality for give the ability to the agent to design and select path planners.

In fact, the ARS model does not contemplate this functionality so the need to create another module to do this task is obvious. Moreover, this module must be designed and implemented thinking not only in the UT2004 but also for a general unknown environment due to this is the final goal of the work of all the people (PhD students, bachelor students and master thesis students, etc.) who work in this research project.

Again, and just as a provisional measure, the navigation of the embodied agent was implemented in the pogamut side because of convenience. It had not have too much sense do it in the interface, to be concrete in the body interface. It is explained before as well, but basically the point is that the embodied agent should not remember every points of the map, at least not all, it must navigation in a "blind" way. In other words, it must follow the goal to don't crash with walls. To do that, ray castings were defined and the agent follows values of them to know when it must move forward or turn.

6.2.3 New Actions and Use-cases

To check all this improvements and to have more scenarios to force the embodied agent live in other conditions, new use cases must be designed. And not only that, it will give the possibility to known

if the new actions that must to be implemented inside the ARS model and selected by the decision unit. It will to permit to make one step forward in the mission to create a humanlike architecture.

A lot of different new stuff must be done related to actions, but a start point could be the completion of move_to action. The point is that: until now, this action does not have any more porpoise than picking up a healthpack. It seems logical that an agent needs more things in a war environment, for instance it should know when its weapons do not have bullets, so in this scenario the agent should go to pick up them. And not only bullets, also armor and new weapons. Talking about new weapons, a prioritization of the weapons should be done due to not all have the same features and, because of that, not all are proper in a particular situation. One consequence of this would be the definition of an ARS object for each staff that would be wanted to pick up.

A new action, that does not have the need to be related to an object, but is related to other agents, is shoot. Although this action is used in one of use cases defined in this work, it is a functionality that is cover by the interface. In a future, this action should be created in ARS model and give resources to the decision unit to selected when the agent sees an enemy and it has an enough health to fight.

Another interesting scenario could be to make the agent one enemy, while it is fleeing. This use case will give be a good tester for the vision system and for the navigation of the agent as well. Apart from create the action follow, a huge impact from these two new improvements will be required and in that moment a serious implementation will be done. It will be close to a human.

6.2.4 Improve ARS Speed Processing

During the work of this work, another weak of the ARS model that have been detected was the speed processing of it. The UT2004 bot, that are playing the role of ARS embodied agent, need a speed processing really fast. Unfortunately, ARS decision unit is not fast enough to satisfy this requirements form UT2004 environment and because of that the agent must wait every loop to receive the action.

To improve it, a different computational system is proposed. Until now, the ARS model is processed in a sequential way. It means that all the modules are processed separately, one by one, in order from the first until the last one. One possible solution could be a parallel processing based on different threads. For instance, the drive part could be processed at the same time than the perception part where the Super ego takes part, just before to arrive to change from the primary process to secondary process.

6.2.5 BotPrize

When all these points explained in previous subsections will be done, a good practice to test the ARS model as a mind of UT2004 bot would be to take part in a competition related to this field. For instance, one of these competitions is the BotPrize. Apart from the possibility to earn the prize if the ARS model would provide the most humanlike behavior, it will give the possibility to compete against other decision making architecture based on other models (probabilistic algorithms, Damasio theories, etc.).

In particular, the BotPrize competition challenges researchers to create a bot for UT2004 that can fool opponents into thinking it is another human player. The competition has different sponsors and the best bot is awarded following different rules to test the humanlike behavior. Obviously, to do that the ARS model should be more developed in order to be able to compete.

6.2.6 Abstraction of the Interface and Test it in new Environments

The main goal of this work was to create an interface to be able to connect UT2004 with ARS model. The point was not "contaminate" the ARS model with a specific stuff from the environment that wants to be mated, in this case UT2004. So it gives the possibility to show how the ARS model can be adapted to another environment as a first entity implemented following this porpoise.

Although this interface is as much independent as it was possible, it is not enough abstract. In other words, it is focused on the connection between ARS model and UT2004 and don't give the possibility to connect ARS to another environment. For that reason, a high level interface should be implemented to give resources required to create automatically the action that will be necessary to for a particular environment or the homeostatic parameters for a particular embodied agent.

Independently if this high level interface is created or not, a good practice would be to test the ARS model in other interfaces that the goal of the embodied agent would be different. In the case of the UT2004 the agent goal is to survive in a world were continuously is in danger to be killed or, at least, hurt. One of these new environments could be Emohawk that is a world designed for show the human abilities of an embodied agent. Another Pogamut tool is created to help programmers to integrate the agent into Emohawk world. In fact, Actions will handle everything the agent will perform in the environment, starting with locomotion and ending with conversation. It allows us to have more control over agent behaviour. For instance, a depressed agent will walk more slowly and with different animation, etc.

Literature

[Ahm12] S. Ahmed: Sigmund Freud psychoanalytic theory Oedipus complex: A critical study with reference to D. H. Lawrence "Sons and Lovers", Internal journal of English and literature Vol. 3(3), pp. 60-70, March 2012.

[Boe97] G. Boeree: SIGMUND FREUD (1856-1939): Personality Theories, Psychology Department Shippensburg University, pag 30-45 ,1997.

[Deu11] T. Deutsch: Human Bionically Inspired Autonomous Agents, PhD Thesis, Vienna University of Technology, chapters 2-3-4, 2011.

[Dig09] F. Dignum, J. Bradshaw, B.Silverman, W. van Doesburg: Agents for Games and Simulations: Trends in Techniques, Concepts and Design, ISBN-10 3-642-11197-1, Springer, Berlin, 2009, chapter 2.

[Dig11] F. Dignum,: Agents for Games and Simulations II: Trends in Techniques, Concepts and Design, ISBN 978-3-642-18180-1, Springer, Berlin,2011,chapter 1.

[Don09] B. Dönz: Actuators for an Artificial Life Simulation, Diploma Thesis, Vienna University of Technology,,chapter 4, 2009.

[Duc11] A. Duchini, E. Herrero: Desarrollo de agentes inteligentes para videojuegos en primera persona, Master Thesis, PROYECTO DE SISTEMAS INFORMÁTICOS, Universitas Complutensis Matritensis, Madrid ,2011.

[Ein03] Einhorn, Jeffery, M. and Chang-Hyun Jo, A Use-Case Based BDI Agent Software Development Process, Proc. of the 2nd International Workshop on Agent Oriented Methodologies - OOPSLA-2003, Anaheim, CA, USA, Oct.2003.

[Fre33] Sigmund Freud. New Introductory Lectures On Psycho-Analysis., volume Volume XXII (1932-1936): New Introductory Lectures on Psycho-Analysis and OtherWorks, 1-182 of The Standard Edition of the Complete Psychological Works of Sigmund Freud. Hogarth Press and Institute of Psych{Analysis, 1933.

[Gem10] J. Gemrot, R. Kadlec, M. Bída, O. Burkert, R. Píbil, J. Havlíček, L. Zemčák, J. Šimlovič, R. Vansa, M. Štolba, T. Plch, C. Brom,: Pogamut 3 can assist Developers in building AI for their videogame Agents, Faculty of Mathematics and Physics, Charles University in Prague, 2010.

[Goy11] A.Goyal, P.Pasquier: Human-like Bots for Unreal Tournament 2004 A Q-learning Approach to Refine UT2004 Strategies, School of Interactive Arts and Technology, Simon Frase University, Surrey, BC, 2011.

[Huh98] M. Huhns, M. Singh: Reading in agents, Section 3.1: A pragmatic BDI architecture, Morgan Kaufmann Publishers, San Francisco, USA, 1998.

[Lai12] E. Laird, B. Congdon: The Soar User Manual, Computer Science and Engineering, University of Michigan, , chapter 2,2012.

[Lan10] R. Lang: A Decision Unit for Autonomous Agents Based on the Theory of Psychoanalysis, PhD Thesis, Vienna University of Technology, chapter 4,2010.

[Mill88] T. Miller, R. Hewes: Real time experiments in neural network based learning control during high speed nonrepetitive robotic operations, Department of Electrical and Computer Engineering Kingsbury Ball University of New Hampshire, whole document,1988, IEEE International Symposium on intelligent control.

[Mun12] E. Muñoz: Implementation of the Artificial Recognition System Decision Unit in a Complex Simulation Environment, Master Thesis, Vienna University of Technology, chapter 4, 2012.

[Ole95] O'Leary: On the history of AI applications, Univ. Of Southern California, Los Angeles, CA , II IEEE Conference Intelligence Applications 1995, whole document.

[Olm11] Olmos, J.: Personajes con razonamiento basado en casos para videojuegos en primera persona, Master Thesis, Departamento de Informática e Ingeniería de Sistemas ,Universidad Zaragoza, 2011.

[Rao95] S. Rao, P. Georgeff: BDI Agents: From Theory to Practice, Australian Artificial Intelligence Institute, Melbourne, Australia, whole document,1995.

[Rid06] J. Ridgway: SIGMUND FREUD (1856-1939), 2221 Theory & Practice 1: Lecture 3, Semester 1, 2006.

[Rob93] P. Robinson: Freud and his critics, University Of California, Los Angeles, 1993.

[Ut04] Unreal Tournament 2004: UT2004 Manual.

[Wied09] Jirí Wiedermann: A high level model of a conscious embodied agent, Academy of Sciences of the Czech Republic , Czech Republic 2009. 8th IEEE International Conference on Cognitive Informatics, whole document,2009.

[Zei10] H. Zeilinger: Bionically Inspired Information Representation for Embodied Software Agents, PhD Thesis, Vienna University of Technology, chapter 2-3-4, 2010.

Internet references

[1] http://ars.ict.tuwien.ac.at/

[2] http://pogamut.cuni.cz/pogamut_files/latest/doc/tutorials/

[3] Unreal Tournament.Epic Games, Inc.Product homepage http://www.unrealtournament.com

[4] http://www.wikipedia.org/dokuwiki

[5] http://www.eclipse.org/org/

[6] http://java.sun.com

[7] http://www.marxists.org/reference/subject/philosophy/works/at/freud2.htm

7. Appendix

7.1 Environment set up

7.1.1 Environment installation

1. Install copy of Unreal Tournament 2004.

2. Install Eclipse 3.6, or 3.7
   - Download Eclipse IDE for Java Development
   - Unpack it to some place, e.g. c:\eclipse
   - Turn off UAC (some plugins are having problems with UAC during installation)
   -

3. Install Subclipse plugin
   - see Subclipse Download&Install web page
   - copy to clipboard correct update site
     - e.g.: http://subclipse.tigris.org/update_1.8.xstartup
   - startup Eclipse, go to Menu - Help - Install New Software
   - click on “Add” button on the right-upper side of the dialog
   - add new update site (name it Subclipse and provide link from the clipboard from the Subclipse site)
   - list your new site from the combobox
   - install Subclipse plugins

4. Install M2Eclipse plugin
   - similar operation to installing Subclipse but different update site
   - see m2eclipse web page, particulary m2eclipse Download section
     - pick their update site,
       e.g.: http://download.eclipse.org/technology/m2e/releases
   - install plugins from their update site

5. Now it is advised to use Tortoise SVN (or some other SVN client) to checkout desired Pogamut project from the SVN
   - Use svn://artemis.ms.mff.cuni.cz/pogamut/project for all projects

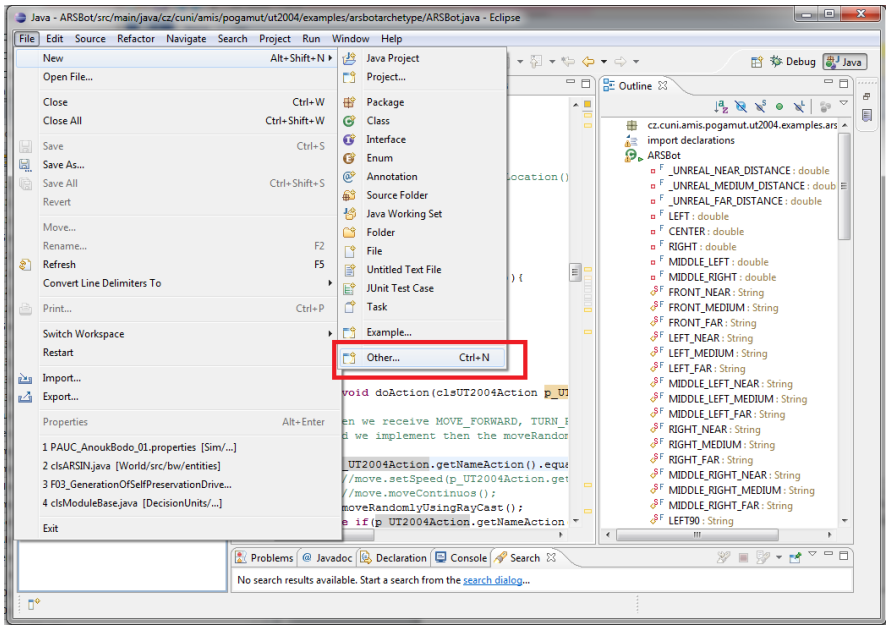
6. Install Unreal Engine 2 Runtime

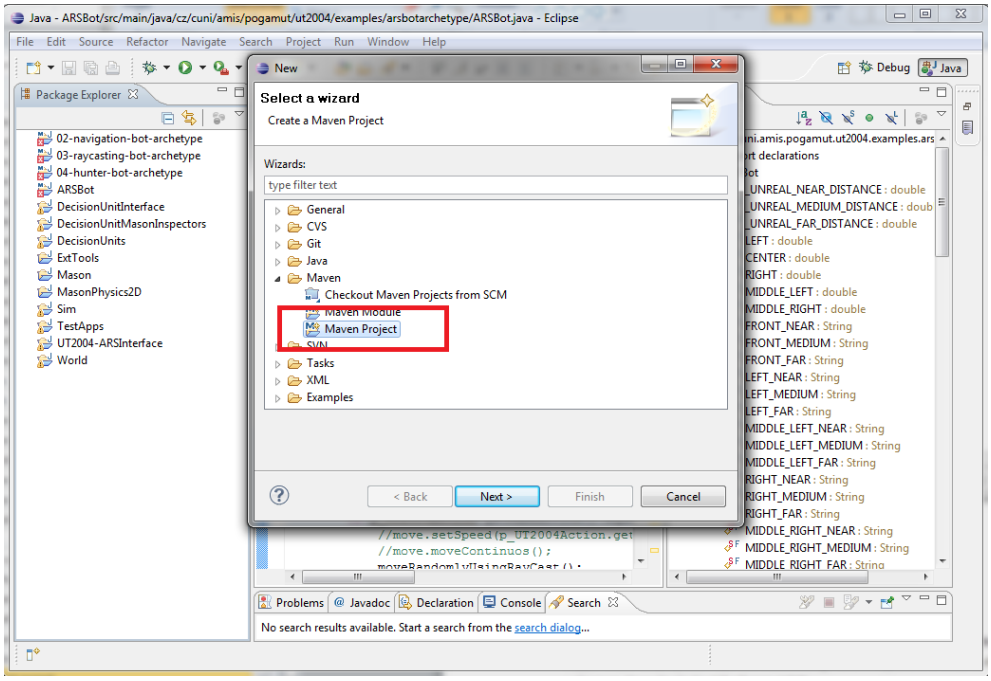
7. Install Pogamut
   - Follow the default selections

7.1.2 Bot project

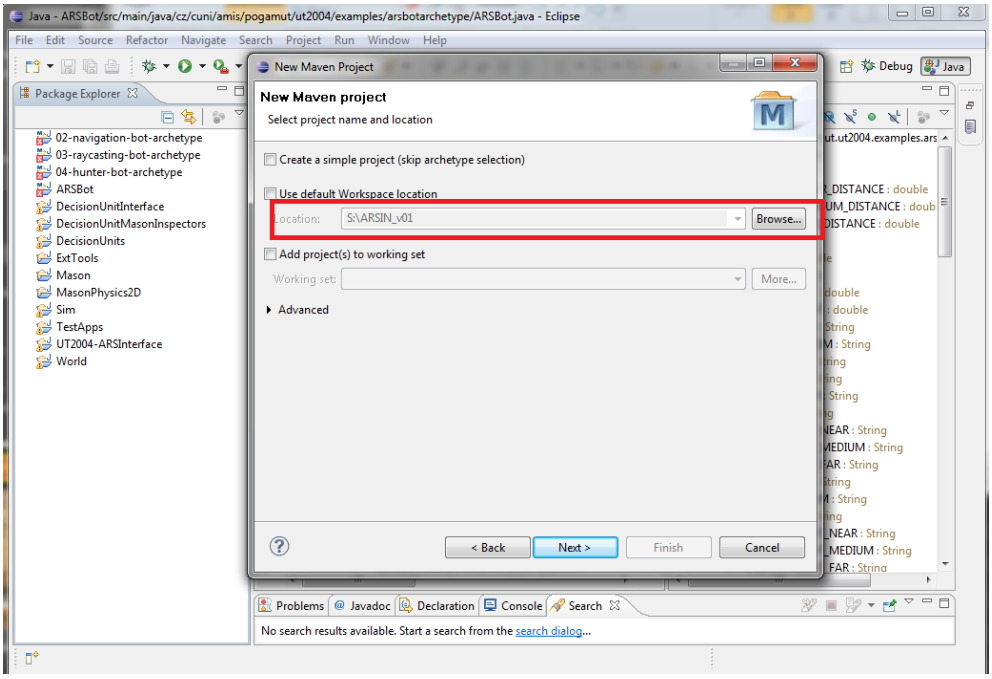
Once the installation steps are complete, two tasks remains: import a Pogamut Bot in eclipse and to create a UT2004 server. In this subsection the first task will be explained and detailed.

1. Frist of all a maven project must be imported.





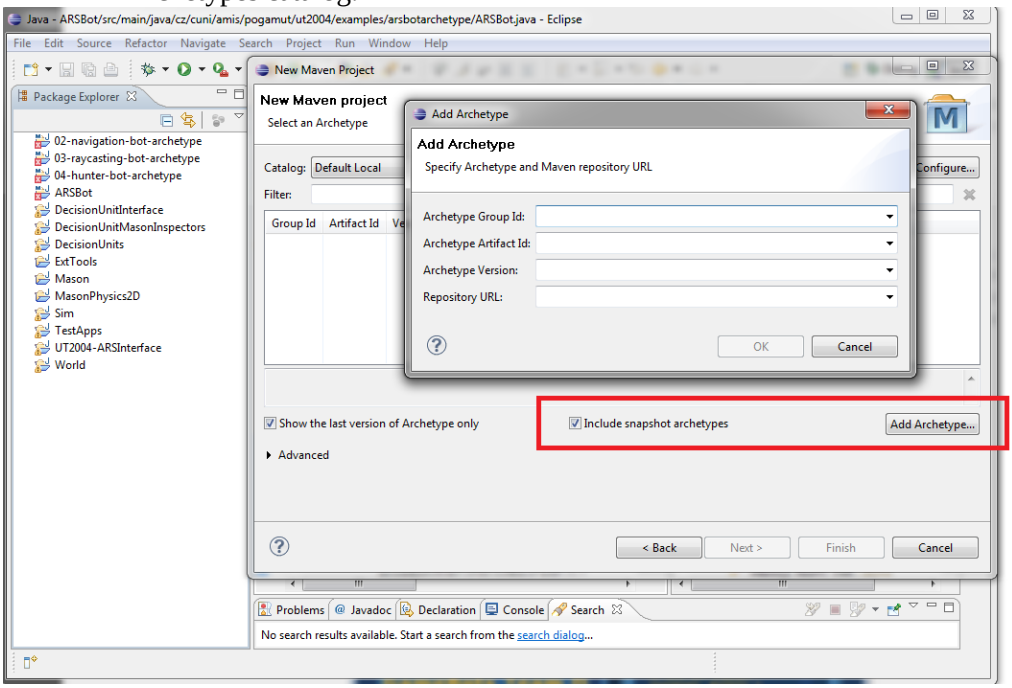
2. Select workspace

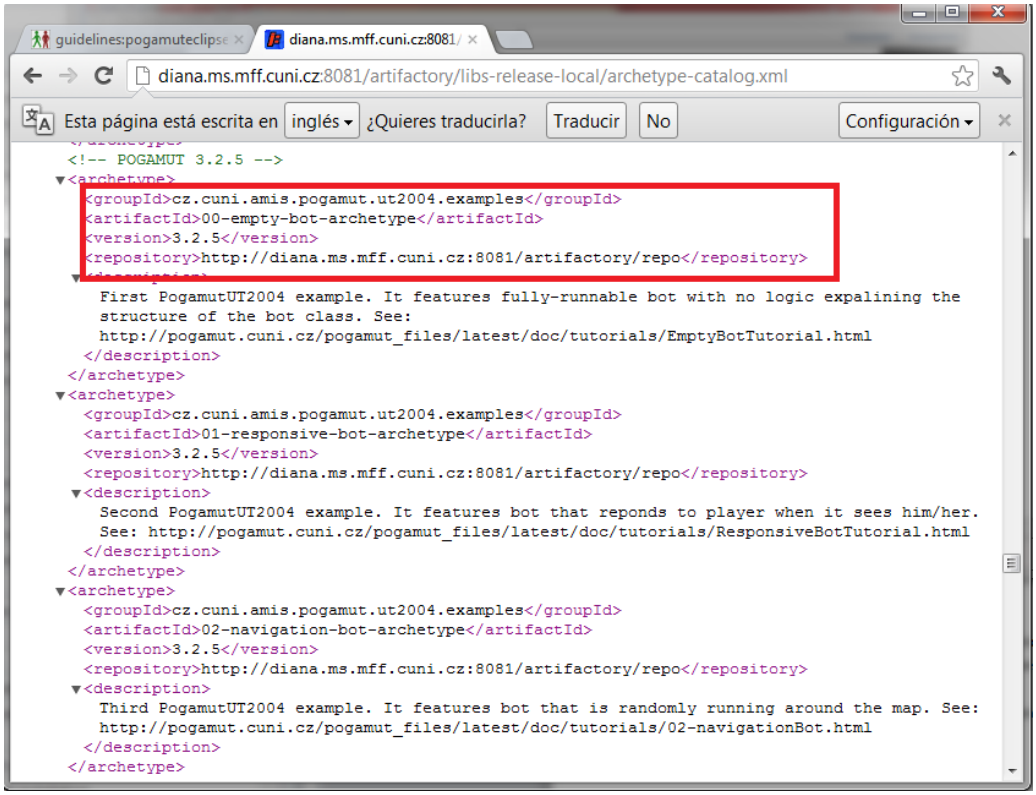


3. Introduces properties from selected bot.

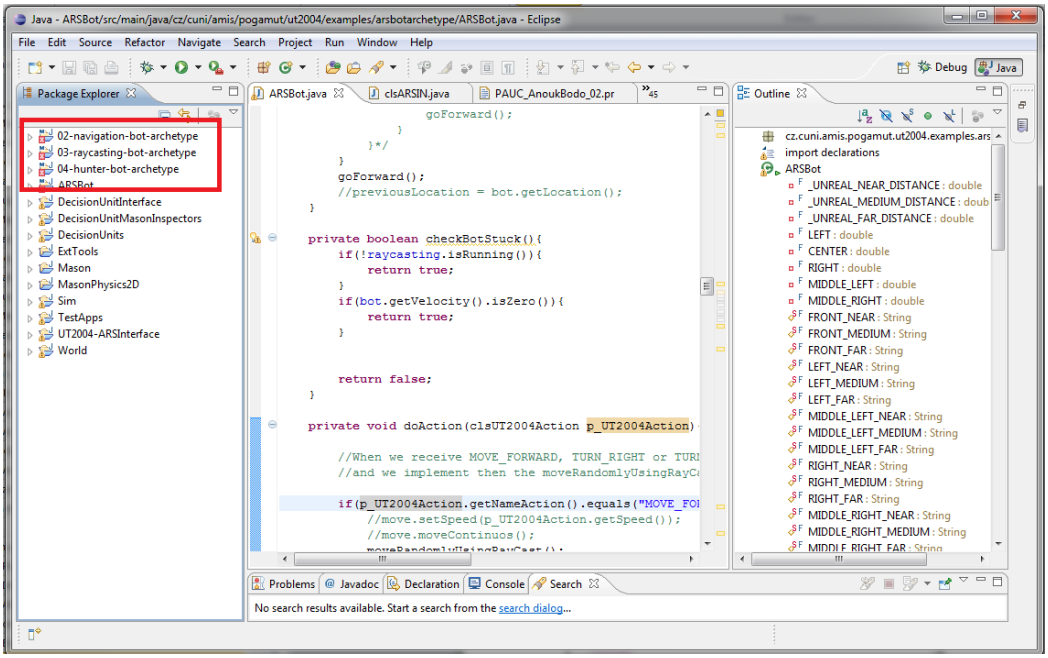
    a. Checks the checkbox include snapshot archetypes and click add archetypes.

b. You may select that one you want to import and introduce the proper information from the up-to-date list of available archetypes in Pogamut Maven archetypes catalog.





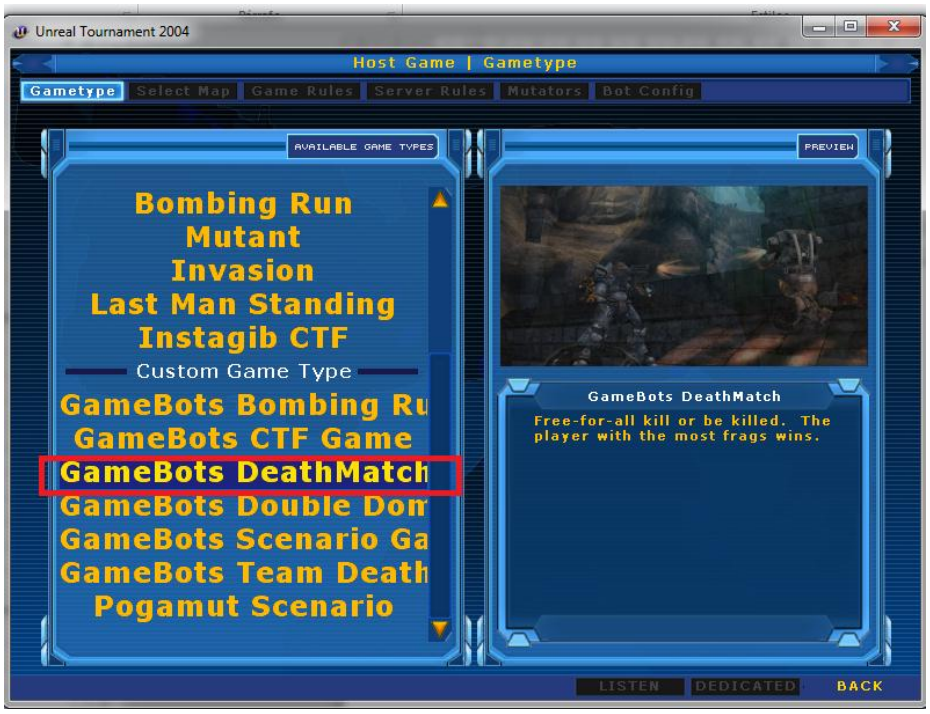
4. Select the new Maven project that has been introduced in the previous step and automatically it will appear in the package explorer. Make a refresh to correct errors.



7.1.3 UT2004 server

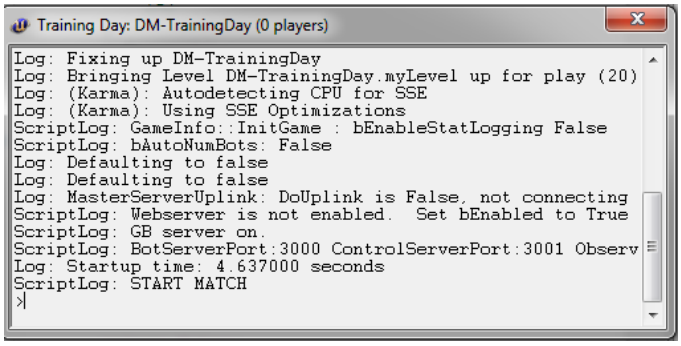
1. Select the option host game.

2. Select the gameBots map desired, in this case death match



3. Select the desired map, in this case training day, clock on *dedicated*.

4. Automatically the server must be created.



5. To Join the server .
    a. Select first join game .

    b. Join the game crated previous.