Imperial College London

Department of Computing

Spiking Neural Networks for Human-like Avatar Control in a Simulated Environment

by

Zafeirios Fountas [zf509]

Submitted in partial fulfilment of the requirements for the MSc Degree in Computing Science of Imperial College London

September 2011

Abstract

The primary focus of this project is the creation of an experimental platform for controlling avatars in a game environment using spiking neural networks as well as the development of suitable networks that make these avatars exhibit human-like behaviour. The platform that has been used is the computer game Unreal Tournament 2004 (UT2004), which provides an efficient environment for comparing human and embodied AI behaviour without the cost and difficulty of real humanoid robots.

The first stage was the development of a wrapper for UT2004 that transforms this video game into a artificial intelligent (AI) agent simulator, that can be used to test AI techniques, such as spiking neural networks, to control avatars within its environment. The second stage was the creation and advancement of a system that uses an architecture based on global workspace theory and spiking neurons and the evaluation of this system using the developed software. This biologically-inspired approach is the first neural implementation of a global workspace architecture that is embodied in a dynamic real time environment.

The conclusion to the project was the participation in the 2K BotPrize human-like bot competition at CIG 2011 in Seoul, South Korea where the system was tested against other approaches and and came a close second.

Acknowledgements

I would like to express my gratitude to my supervisors Dr David Gamez and Dr Andreas Fidjeland, for the limitless support at every stage of this project. The project would have never reached this point without their guidance, inspiration and enthusiasm.
Also, I would like to thank my family and my friends for the love, understanding and support they have shown during the past 12 months of my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Since the beginning of artificial intelligence, a large repertoire of software systems have been designed to behave in a similar way to humans. In 1950 Turing [36] proposed a method that could be used to judge the performance of such systems. In his famous test, a human judge carries out two conversations over a text-only channel. The system that is to be judged participates in one conversation and imitates the behaviour of a human talker while a real human carries out the second conversation. Then, the judge has to decide which is the real human.

Thus far, a substantial number of chatting software systems have been proposed, which have tried to pass this test without success. This can be partially ascribed to the fact that the majority of these programs rely on conversational tricks when they cannot understand the semantics of the judges' questions. A second reason is the lack of physical embodiment which makes the programs unable to process questions related to their physical existence in the real world.

A significant number of variations on the Turing test have been proposed since 1950, with the majority emphasising embodied systems. One of these variations is the the 2K BotPrize [1] competition that was proposed by Hingston [20] and sponsored by 2K Australia. This competition evaluates the performance of software systems that try to control an avatar in the computer game Unreal Tournament 2004 (UT2004) in a human-like manner. Competing software agents and human players direct the avatars and human judges decide whether the avatars are controlled by humans or artificial systems. The main goal of this competition is to promote the creation of a software system that will be indistinguishable from the human players. Then, according to Turing [36], this system might be considered to have human-level intelligence within the environment of this game. In 2010 the competition was won by a system based on a model of consciousness [2], more precisely on an implementation [27] of the global workspace theory.

Global workspace theory(GWT) is a simple and popular cognitive architecture that was proposed by Baars [8] and is claimed to be a model of consciousness as it has successfully modelled different characteristics of consciousness. A major advantage of this theory is that its assumptions regarding the operation of human brains are consistent with our current knowledge of the cortex. For these reasons, a number of research projects in neuroscience have used neural implementations of GWT to investigate how it might be biologically implemented [32, 13]. Also, GWT has been successfully used to control complex software systems carrying out different tasks. However, so far, only programmatic implementations have used GWT to solve challenging problems in real time.

Furthermore, spiking neural networks are biologically-inspired networks that model the behaviour of neurons in the brain. Such networks have a number of interesting properties when compared with the rate-based models found in the traditional artificial neural network literature, with their main difference coming from the incorporation of the concept of time. For example, in these networks, discrete spikes and propagation delays can be used to identify temporal patterns. One great advantage of working with such biological approximations of real neurons instead of using traditional neural networks is that this could lead to a better understanding of how the brain works. Also, biologically-inspired neural networks could help us to develop more intelligent machines by imitating cognitive functions of the brain. Our ability to simulate spiking neural networks has greatly improved recently, and the Department of Computing of Imperial College London has a high-performance simulator running on a cluster equipped with CUDA-based graphics processing units (GPUs) [3].

1.2 Objectives

This project aims to show that biologically inspired neural mechanisms can be an effective way of constructing a software system that exhibits with human-like behaviour. For this reason, the project also aims to develop a neural architecture that is based on spiking neural networks and global workspace theory as well as a software system that uses this architecture to control avatars within the UT2004 in a human-like manner.

A third aim of the project is to close the gap between the successful programmatic implementations of GWT and the models that have been developed in neuroscience. Achieving this could also help to clarify how a global workspace might be implemented in the human brain.

The completion of the project was divided into a number of milestones. The first stage was the creation of a wrapper for UT2004 as well as an environment in which all of the data from this wrapper is available and it is general enough to be capable of simulating the behaviour of multi-agent systems exploiting UT2004's features. The next stage was the addition to this program of the ability to support spiking neural network (SNN) architectures by decoding the input and output data of the framework into spike patterns and integrating the NeMo spiking neural network simulation environment [14].

The third stage of the project was the actual implementation of a neurally controlled agent that exhibits human-like behaviour within the game environment. For the high-level coordination of the system, the design of the neural architecture was based on global workspace theory and more specifically on a model proposed by Shanahan [32]. To achieve this implementation, a number of simpler neural systems were developed and formed the basis of the final neural architecture.

The final neural system integrated the previous work and became capable of a complete exploitation of GameBots features. When controlled by this system, the avatar was able to defend itself from other players or software agents and act completely independently.

The final step of the project was the evaluation of the ability of the system to exhibit human-like behaviour. This was achieved using two different methods. Firstly, a number of matches were held in order to record statistical data of human players and other software agents and then compare their behaviour and performance. Finally, the system participated in the BotPrize 2011 competition, which can be considered as the ideal test that indicates the level of humanness of the system.

1.3 Report Structure

The rest of this report is divided into 5 chapters. First, Chapter 2 provides definitions of spiking neural networks and global workspace theory as well as justification about why these methods were chosen. Also, it includes references to work that has been considered related to this project and has influenced its development and it closes with a brief description of the software tools that have been used.

Next, chapter 3 provides a complete description of the neuronal architecture that has been designed and used to control an avatar within UT2004 along with justification about the design choices. Furthermore, Chapter 4 describes the software that have been developed in order to simulate and test a system that includes the aforementioned neuronal architecture. Again, justification about the design choices is also included.

Also, Chapter 5 describes two methods that have been used to evaluate the performance of this system along with the outcome of this evaluation. Thereafter, Chapter 6 gives the conclusion of this project and includes a list of the achievements, the next steps as well as a number of potential applications. Finally, Appendix A provides tables with all available data that have been used for the evaluation of the system while Appendix B provides information for the final user of the developed software.

Chapter 2

Background

This chapter provides an overview of the background knowledge that has been taken into account during the design of the system of the present project. The first part is a brief description of artificial neural networks with an emphasis on spiking neural networks. Thereafter, a brief introduction to the term cognitive architecture precedes the description of the global workspace theory, the cognitive architecture that the present work has been based on. Lastly, comes a brief review of relevant published work that has been taken into account as well as the software tools that have been used.

2.1 Spiking Neural Networks

Biological neurons, or nerve cells, are electrically excitable specialized cells that constitute the biggest part of the most advanced and complex organ in a mammalian body, the brain. Neurons mainly consist of three parts which are the cell body (or soma), the axon and dendrites (see Figure 2.1). Dendrites carry incoming electrical signals originating from other neurons while the axons deliver electrical signals to other individual target cells. A single neuron can be connected to as many as 10,000 other neurons. The junction that permits two neurons to exchange electrical signals is called synapse. One of the most remarkable characteristics of neurons is their ability to propagate electrical signals very fast and over very large distances. They achieve this by rapidly changing the difference in voltage between the interior and exterior parts, i.e. their membrane potential, which results the generation of electrical pulses that are called action potential or spikes.

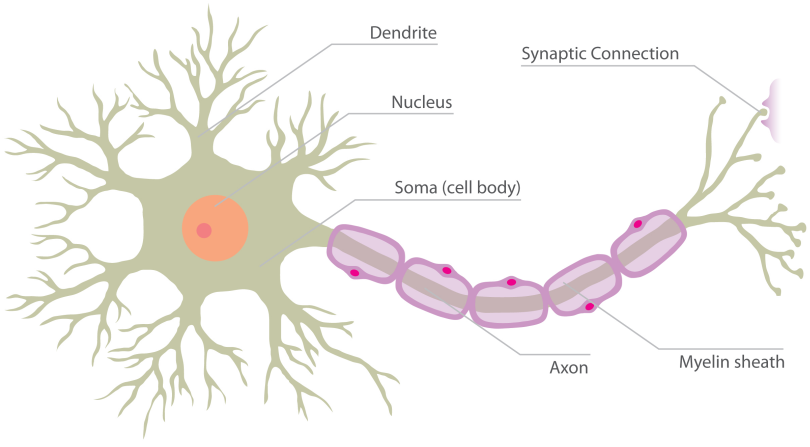


Figure 2.1: The structure of a real neuron.

Artificial Neural Networks (ANN) are mathematical models inspired by the biological neural networks. Classical ANN models consist of mathematical substitutes of the processing elements of the real neural networks such as neurons, axons, dendrites and synapses (see Figure 2.2) as well as a specific topology i.e. the network architecture that connects the various types of neurons.

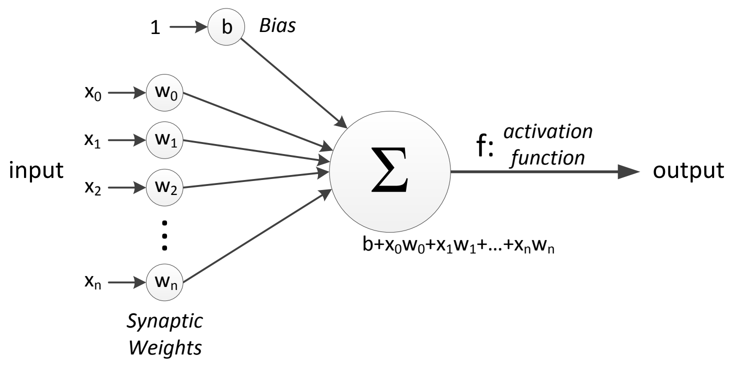


Figure 2.2: The simplest mathematical model of a neuron called the Perceptron. [30]

The wide impact of ANN systems comes from their ability to learn. According to Mitchell [5], “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improve by experience E.” Likewise, ANNs can be configured in a way that the application of an input dataset results in the corresponding desired output. More specifically, the process of learning (or training) in classical ANNs can be described as the repeated adjustment of the synaptic weights between the neurons, according to a learning rule, while the system is being fed with input data, whose desired output is known. Then, the process terminates when the error between desired and the real output is acceptable.

Models of ANNs have been very useful in machine learning, where the solution of a problem lies in learning an approximation of a target function using large sets of data. This includes problems of classification, prediction, and pattern recognition. Furthermore, another significant feature of ANNs is that the process of the work is performed collectively and in parallel by the neurons which makes them particularly useful in systems that can support parallel processing.

Because of their good performance in the aforementioned types of problems, ANNs have been applied to various domains such as robot control, sound/image recognition, emotion recognition and stock market prediction.

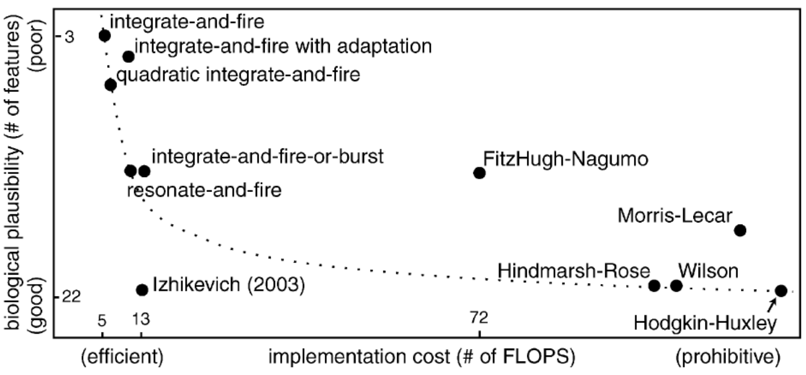
Although classical ANNs have helped artificial intelligence in various domains, they are far away from being capable of simulating the complex behaviour of mammalian brains. In order to achieve that, more realistic approaches had to be used that could effectively simulate the dynamics of the real neurons. In order to solve this problem, a new category of ANNs the so-called spiking neural networks was introduced [21].

Spiking neural networks (SNNs), are often referred to as the third generation of ANNs [25] following the Binary Neural Network models, where the activation of the neurons is binary and Real-valued NNs, where the 'mean firing rate' of a neuron is considered. The main addition of SNNs to the existed models was the incorporation of the concept of time. This makes the simulation richer and more biologically natural, allowing a variety of additional concepts to be included, such as spike latencies, axonal conduction delays and partially synchronized network oscillations.

According to the work of Maass [25], SNNs are computationally more powerful than classical ANNs which means that a classical ANN needs more neurons than a network of spiking neurons to be able to handle a specific problem.

One of the most important factors in designing a system based on SNNs is the neuron model that is going to be used. There is a large repertoire of different choices that have been proposed, varying from models that behave impressively similar to real neurons, to more simplified models that are less biophysically accurate but also much cheaper in terms of required computational resources.

Figure 2.3 presents a comparison between the biological plausibility and the computational needs of the most popular models as well as their basic properties.



Models	biophysically meaningful	tonic spiking	phasic spiking	tonic bursting	phasic bursting	mixed mode	spike frequency adaptation	class 1 excitable	class 2 excitable	spike latency	subthreshold oscillations	resonator	integrator	rebound spike	rebound burst	threshold variability	bistability	DAP	accommodation	inhibition-indiced spiking	inhibition-induced bursting	chaos	# of FLOPS
integrate-and-fire	-	+	-	-	-	-	-	+	-	-	-	-	+	-	-	-	-	-	-	-	-	-	5
integrate-and-fire with adapt.	-	+	-	-	-	-	+	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	10
integrate-and-fire-or-burst	-	+	+		+	-	+	+	-	-	-	-	+	+	+	-	+	+	-	-	-		13
resonate-and-fire	-	+	+	-	-	-	-	+	+	-	+	+	+	+	-	-	+	+	+	-	-	+	10
quadratic integrate-and-fire	-	+	-	-	-	-	-	+	-	+	-	-	+	-	-	+	+	-	-	-	-	-	7
Izhikevich (2003)	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	13
FitzHugh-Nagumo	-	+	+	-		-	-	+	-	+	+	+	-	+	-	+	+	-	+	+	-	-	72
Hindmarsh-Rose	-	+	+	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+		+	120
Morris-Lecar	+	+	+	-		-	-	+	+	+	+	+	+	+		+	+	-	+	+	-	-	600
Wilson	-	+	+	+			+	+	+	+	+	+	+	+	+	+		+	+				180
Hodgkin-Huxley	+	+	+	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+		+	1200

Figure 2.3: Comparison of the neuro-computational properties of spiking and bursting models; "# of FLOPS" is an approximate number of floating point operations (addition, multiplication, etc.) needed to simulate the model during a 1 ms time span. Each empty square indicates the property that the model should exhibit in principle (in theory) if the parameters are chosen appropriately, but the author failed to find the parameters within a reasonable period of time. (Figure and description adapted from [23])

Hodgkin-Huxley Model

In 1952, Alan Hodgkin and Andrew Huxley proposed a mathematical model [21] that consists of a set of non-linear ordinary differential equations (ODEs) and describes the propagation of the action potentials in neurons. As shown in the equation 2.1 from [21], the model considers the potential of a neuron as a function of currents of ions that flow across its membrane in addition to the current that the neuron receives from the dendrites. The ionic currents that are taken into account are the potassium, the sodium and the leakage currents and are expressed in the following equation with the term I_i. The leakage current parameter represents all the rest insignificant ions which cross the membrane.

$$I = C_M \frac{dV}{dt} + I_i \tag{2.1}$$

where

I	is the total membrane current density,
I_i	is the ionic current density,
V	is the displacement of the membrane potential from its resting value,
C_M	is the membrane capacity per unit area and
t	is time.

Although Hodgkin-Huxley is one of the oldest mathematical spiking neuron models, it is able to mimic the behaviour of different types of neurons very accurately. However, as shown in Figure 2.3, a significant drawback is its excessive computational cost. For the great importance of their work, Hodgkin and Huxley received the Nobel Prize in Physiology or Medicine in 1963.

Izhikevich Model

Another popular model that consist of a fairly good compromise between computational efficiency and biologically realistic behaviours is the model proposed by Izhikevich [22].

This model is able mimic the dynamics of the different types of neocortical neurons almost as well as the Hodgkin-Huxley model and it supports the majority of the features shown in the table of Figure 2.3 which makes it one of the most biologically plausible choices. The only transparent weakness of Izhikevich model is the lack of biophysical meaning of the concepts used in the model.

The Izhikevich model is governed by two differential equations.

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \tag{2.2}$$

$$\frac{du}{dt} = a(bv - u) \tag{2.3}$$

where v is the membrane potential, u is a recovery variable, I is the dendritic current, and a and b are abstract parameters of the model.

When the voltage exceeds a threshold value (preset at 30) then v and u change values according to the following

$$u \rightarrow c$$

$$u \rightarrow u + d$$

Hence, the complete model includes four parameters. According to Izhikevich [23], "The model can exhibit firing patterns of all known types of cortical neurons with the choice of parameters a, b, c, and d given in [22]."

The Izhikevich model has been used in numerous applications of Computational Neuroscience research such as the Global Neuronal Workspace model of Shanahan [32] and the NeMo spiking neural network simulation environment [14]. Finally, this model was used for the simulation of spiking neurons in the present work.

2.2 Global Workspace Theory

Cognitive Architectures are commonly used among artificial intelligence (AI) researchers in order to develop software agents that show some form of intelligent behaviour. In the first pages of his book [6], psychologist John Anderson gives a definition for the term cognitive architecture: "Cognitive architectures are relatively complete proposals about the structure of human cognition.". He also mentions that a cognitive architecture tries to provide a complete specification of a system following a certain abstractness. As implied here, usually, cognitive architectures do not cover all the aspects of human cognition. Contrariwise, some of these models tend to emphasise in specific aspects such as consciousness.

Models of consciousness are theoretical models that link brain physical (neural) properties to phenomenal properties of consciousness. Although a large number of such models have been proposed so far and can be found in the literature, there is not a single model at present that can sufficiently explain and combine all the different aspects of the conscious experience. Also, the level of abstraction of those models vary significantly since only a few of them propose mechanistic implementations that make them computationally feasible in order to be used for the development of intelligent agents. One of the most widely used models among computational neuroscientists and AI researchers is Baars global workspace theory [8] and its neuronal implementation called Global Neuronal Workspace [34, 32].

Global workspace theory (GWT) is a simple and widely used cognitive architecture that consists of a model of consciousness and it was first suggested by Baars [8, 9, 7]. According to this theory, the most competitive part of cognitive content becomes conscious, which means that it becomes globally accessible to the rest parallel unconscious cognitive processes (see Figure 2.4). To support that view, GWT relies on the assumption that a large part of the primate brain consists of highly specialized regions. This can be considered consistent with the current human knowledge of the brain.

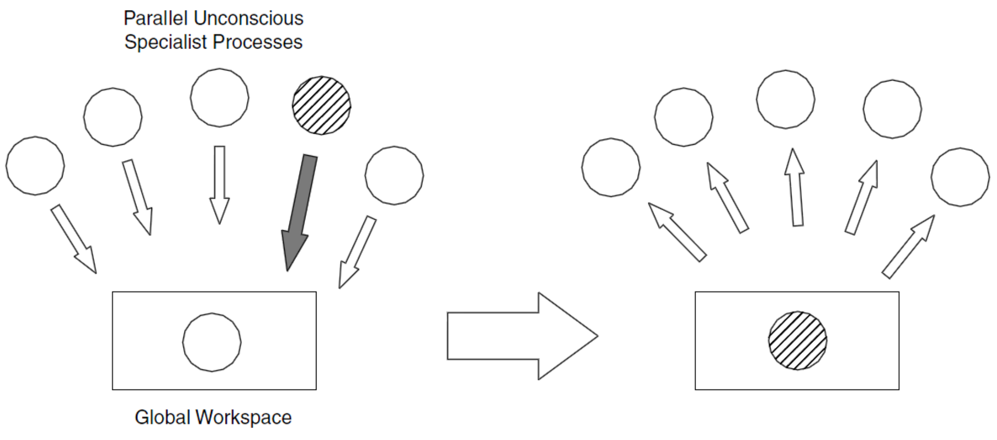


Figure 2.4: The basic global workspace architecture (from [31]).

The model of GWT was originally described using a theatre metaphor. In this "theatre of consciousness", there is a stage, which corresponds to the working memory, in which players are moving in and out making speeches and competing (or sometimes cooperating) in order to be in the spotlight. The players are compared to a number of processors in working memory while the spotlight is a mechanism for attention which reveals the contents of the consciousness. It is worth mentioning that only some of the players are each time in the spotlight. In this theatre there is also the audience which is in the dark, watching the play. The audience is compared to all the other processors of the system that have access to what has become visible by the spotlight and they are only activated when something interesting happens or someone from the cast calls them.

Finally, the last element of the metaphor is the backstage which is also in the dark, though also contributes to the activities that happen in the bright spot.

The backstage corresponds to the contextual systems where each context is a non-conscious coalition of processors that shape conscious contents without ever becoming conscious themselves. Contexts include unconscious expectations and intentions as well as processing of visual information and many others. The players include outer senses, e.g. seeing, hearing, tasting etc, inner senses such as inner speech, visual imagery and dreams, and ideas. Finally, the unconscious audience includes the various memory systems, interpretations of conscious contents such as recognition systems for objects, faces, speech, etc, as well as motivational systems, and various automatisms such as decision making and action selection.

The basic principles of the above architecture make it highly suitable for the advancement of relevant computational models that aim to simulate elements of conscious behaviour, since it considers the nervous system as a distributed parallel system where a number of different specialized processes coexist. GWT has given inspiration to numerous implementations and some of them are presented in the next section of this report.

2.3 Related Work

In this section there is a brief description of different scientific researches that have been considered related or have influenced the design of the current project.

Robotics and Spiking Neurons

According to the current literature, a significant number of research projects have been focused on the control of robots using biologically-inspired neural networks.

In particular, a significant example is the work of Krichmar et al. [24]. As a part of this work, a controller of a wheeled robot (Darwin X) had been developed. This controller consisted of a simulation of a vertebrate nervous system, based on detailed aspects of the anatomy and physiology of regions of the mammalian brain and included models of the visual system and hippocampus. The simulated network consisted of 90,000 rate-based neuronal units and 1.4 million synaptic connections. This system used learning to solve a maze task where the robot is rewarded by finding a hidden platform from any starting position using only proprioceptive sensors and visual landmarks. The simulation of the nervous system was run on a cluster of computers that interfaced with Darwin X through wireless links.

Furthermore, in [17] Gamez has developed a neuronal controller that directs the eye movements of a simulated robot towards positive stimuli based on reasoning about future actions. In this work, the network consisted of approximately 18,000 neurons and 700,000 synapses and the architecture was based on a model of consciousness called Information Integration Theory of Consciousness [35]. This architecture also incorporated the concepts of emotion and imagination, allowing the robot to have a different response to stimuli of different colours and to imagine the result of different movements respectively.

Another worth mentioning example is the work of Bouganis and Shanahan [10]. This work concerns the design and implementation of a trained spiking neural network which is able to control four degree-of-freedom robotic arms. This system is being trained based on a rule of Spike Timing-Dependent Plasticity (STDP). It has been tested that after an initial period of motor babbling it is able to successfully control the four-joint robotic arm of an iCub humanoid robot.

Finally, in [19], Hagras et al presented a SNN controller for mobile robots as well as an adaptive on-line genetic algorithm (GA) that is used to train the aforementioned network in real time through interaction with the environment.

Programmatic implementations of global workspace

Different versions of a global workspace have been successfully adapted to control the information flow in complex systems. In [16] Franklin developed IDA a naval dispatching system that is considered to be functionally conscious, to negotiate new duties with US Navy sailors. More specifically, this system was aimed to match individual sailors to new navy assignments based on the sailor preferences and abilities and the Navy requirements. The interaction with the sailors was done using natural language conversation. The system had also access to Navy databases in order to process all the information and do the best match.

To perform this task, the system was organized into a large number of specialized codelets using a computational model of GWT. According to Franklin [16], "Codelet is a special purpose, relatively independent, mini-agent typically implemented as a small piece of code running as a separate thread." In [37], an improved version of IDA was introduced. The new model called LIDA (or Learning IDA), among other new features, included several modes and styles of learning.

Another successful approach which is closer to the current project is the work of Arabales et al [27]. This work concerns a general-purpose cognitive architecture aimed to control robots and based on GWT. The system that has been developed to support this architecture consists of two components. The first is called CERA and it is a layered control architecture that uses services provided by the second component, CRANIUM, which is a tool that allows the manipulation of parallel processes and organizes them using two shared workspaces. This system has won the Bot-Prize in 2010 [2] and the human-like bots competition at the 2011 IEEE Congress on Evolutionary Computation.

Neuronal architectures using global workspace

Finally, there is a number scientific researches that have been focused on neural models of GWT in order to investigate the potential existence and operation of this architecture in the human brain.

To begin with, a brain-inspired cognitive architecture that controlled a simulated wheeled robot has been introduced by Shanahan [31]. This work was also based on a model of GWT and among other concepts was focused on the concepts of consciousness, imagination and emotion and the the interplay between them.

Furthermore, Dehaene et al [34, 13, 12] have introduced and advanced a neuronal model that incorporates a global workspace to control the flow of information. Using their model, the so-called neuronal global workspace (NGW), Dehaene and his colleagues investigated how specialised processes interact with the workspace during a psychological paradigm, the attentional blink [28], and while their system was learning the Stroop test [26]. In addition, a related system was proposed by Zylberberg et al. [39].

Although the work of Dahaene and his colleagues has introduced the first neuronal implementation of a global workspace, the primary focus was only on the competition of the cortical processes in order to win access to the workspace, and not in the entire cycle of the information flow as defined in the GWT. In a more recent approach by Shanahan [32], which can be considered as a continuation to the aforementioned work, the mechanics and dynamics of broadcast have also been modelled. In an even more recent work by Shanahan [33], a new hypothesis has been examined according to which neural synchronization is the mechanism implementing the global workspace in the brain.

The programmatic implementations that are described in this section have been used in challenging real-time environments, whereas the systems that use neuronal architectures based on GWT were focused on modelling and examining potential implementations of global workspace in the human brain.

Moreover, the author of this report was not able to find previous work that uses a neuronal model of global workspace to control software agents in video games.

2.4 Simulation Environment and Tools

2.4.1 Unreal Tournament 2004

Unreal Tournament 2004 (UT2004) is a futuristic video game co-developed by Epic Games and Digital Extremes as a sequel of UT2003 and the original Unreal Tournament. It falls into a category of video games known as first-person shooters (FPS), where all real time players exist in a 3D virtual world (see Figure 2.5) with simulated physics and a variety of tools that give the players additional abilities. As implied by the term first person, the senses of every player are limited by their location, bearings, and occlusion within the virtual world.

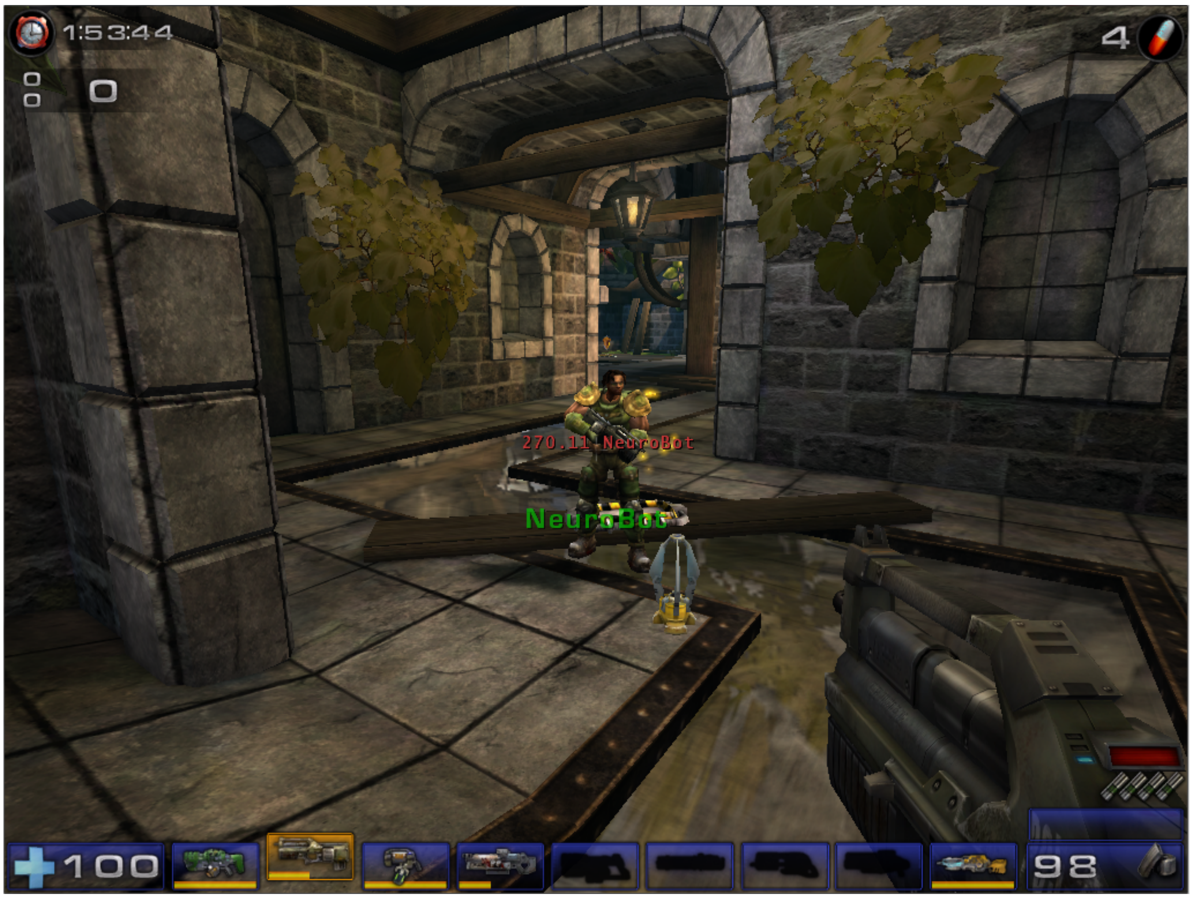


Figure 2.5: Screenshoot taken from Unreal Tournament 2004 ('Deathmatch' mode)

Like its predecessors, UT2004 was designed mostly as an head-to-head arena FPS, the so called 'Deathmatch' mode. However, it also includes two more core game types 'Capture the Flag' and 'Domination' as well as less emphasized modes and a significant number of maps. Also, active online Unreal Tournament's community has been consistently creating new maps and game types. Furthermore, UT2004 consist of fast, dynamic and complex 3D simulation engine, widely available at a very small cost and with a large user base. As a consequence of its popularity, UT2004 has been being tested by hundreds of thousands of people.

Finally, another major feature of UT2004 is that it comes with its own scripting language, UnrealScript [4], which makes it very suitable for modifications. UnrealScript is based in C++ and is responsible for all the game logic and object interaction leaving all the hardcore work such as simulating physics and rendering scenes to the UT2004 game engine. Through this integrated scripting environment, developers are provided with a variety of ways for adjusting physics parameters, developing new game types and world objects or extending existed ones.

Exploiting the above features gives artificial intelligence (AI) researchers a wealth of robust environments for testing their agents. UT2004 environment is also good for students who want to explore agents and AI techniques. To support the above, a new type of research infrastructure was developed, an UnrealScript-based plug-in for UT2004 named GameBots [29].

2.4.2 GameBots

Gamebots is a project that started at the University of Southern Californias Information Sciences Institute, and jointly developed by Carnegie Mellon University. The aim of this project was to turn UT2004 into a domain for research in AI and Multi-Agent Systems (MAS). The main parts of GameBots are a dynamic game engine, which is commercially developed along with some extensions that provide capabilities that are essential for research in the discipline of AI.

The most important features that are provided with GameBots are listed below:

- Support of multiple tasks. GameBots supports a variety of different tasks that concern multi-agent competition such as Capture the Flag and King of the Hill derivatives.
- Extendability. GameBots includes a build-in scripting language that allow researchers that are interested in long-term research to change or extend the supported tasks.
- Easy creation of new environments.
- Human support. One of the most important features of GameBots is that it allows human controlled entities to interact with the AI agents. Hence, it is possible for GameBots to be used for study of human problem solving and scenarios that concern human vs AI and human-AI collaboration.
- It is an open source project and publicly available worldwide.

The above features were characterized as very important, in terms of AI research, mainly because there were not available with other proposed test-bets.

The core of the GameBots 2004 system is a plug-in of Unreal Tournament 2004 that handles the communication between characters in the game and bot clients, via network sockets and TCP/IP protocol. (see Figure 2.6).

As the first step of the GameBots - client communication, the server sends sensory data for the visible world over the network connections. When this data is processed by the client and the next actions are decided, the latter sends back commands and GameBots undertakes to apply the desired actions to the character (e.g. move, shoot, talk, etc) To help agents to navigate to the environment and researchers to implement their algorithms more easily, GameBots provides to the connected clients some extra types of information under the corresponding requests, such as information related to path planning or adjustable ray casting.

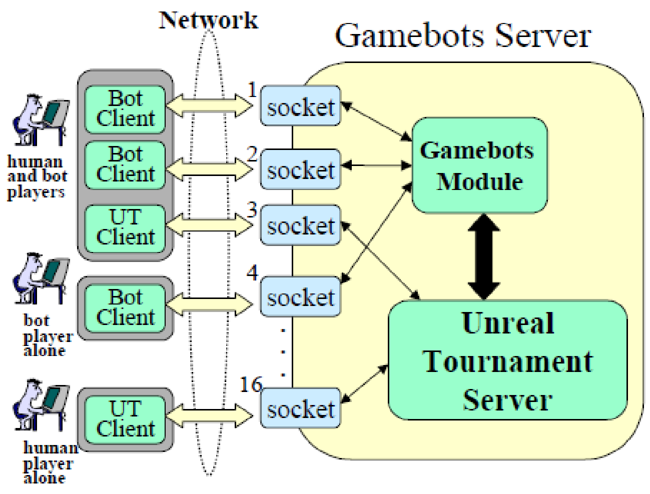


Figure 2.6: Gamebots architecture (human players connect directly to the UT server, and bots connect through the Gamebots Module) (Image adapted from [29]).

GameBots Communication Protocol

Communication with Gamebots is fairly easy since it is able to communicate with any application supports TCP/IP via simple text messages. This is achieved through its interaction protocol for which a detailed description can be found in [29].

After an initial handshake with a client that represents a bot or an observer that watches a bot, GameBots server starts to send synchronous messages every predefined time period. These messages include information regarding the bot's perception. In particular, it includes information regarding proprioceptive sensing such as global position, view direction, velocity, shooting and jumping state, current weapon, etc. The messages also include information about exteroceptive sensing such as positions of visible enemies, distance of walls, positions of visible items and navigation points, etc. Finally, information regarding interoceptive sensing is also included, such as health condition, level of adrenaline and armour condition.

Apart from the synchronous messages, GameBots server also sends a form of asynchronous messages that usually refer to special events that happened in the bot's environment. These events could be for instance hearing a noise, running to a wall, falling, the collection of an item or the death of another player. In addition, they could be responses to client's requests.

The third and final form of messages is the commands. Clients can send messages back to GameBots server asynchronously, and in high frequency. Commands are used to control and configure the bot and it is the only way in which a bot can interact with its environment. A command could be for instance an order to move or look at a specific direction or object, jump or shoot another player.

2.4.3 NeMo

The environment that has been used for the simulation of the neural network of the present system is the NeMo spiking neural network simulation environment [14]. NeMo is a high-performance simulator with the ability to execute simulations in parallel, taking advantage of general-purpose graphic processing units (GPUs). The major scope of NeMo lies in the development and implementation of mathematical models of brain structures in computational neuroscience research.

One iteration of the simulation could be divided into two main phases. The first phase is the update of the states of the neurons based on the equation of the mathematical model used in 1ms time step. The model that NeMo uses is the Izhikevich model that has been described in the previous chapter. The calculations of this phase can be very easily performed in parallel using the available GPU cores. The second phase is the delivery of the spikes that have been produced as an effect of the neuron state update. This phase is more complicated and is considered to be the limiting factor for the performance of the system.

NeMo can be used by applications written in C/C++, Python or Matlab. Using the corresponding application programming interface (API), the controlling application is able to inject external stimulus current into some neurons of the network in order to simulate sensory input, and thereafter convert the output spikes to motor signals. This process can be repeated for each iteration.

The fact that NeMo simulator takes advantage of GPU's parallel capabilities makes it a very reasonable choice for implementing spiking neuron based software agents in a video game environment. The reason is that normally such environments also require hardware that supports parallel processing in GPUs. Hence, hardware compatible with NeMo is typically already included on the final user's platform.

In tests that have been performed using a single device with a high-end consumer-grade GPU, NeMo was able to deliver up to 550 million spikes per second which corresponds to 55 000 neurons in a real time simulation with a biologically plausible number of synapses and neuron conditions [14]. This is fairly adequate for the requirements of the present system which is able to run in real time on a platform that includes a relevant consumer-grade graphics card.

Chapter 3

Neural Architecture Design

This chapter focuses on the design choices that have been made for the final architecture of the neural network as well as a justification for each choice. The architecture that is presented here was initially designed based on the related work that can be found in Chapter 2 and primarily on the GNW architecture proposed in [32]. However, the final version is a result of various reconfigurations during the implementation phase.

The network consists of approximately 20.000 neurons which are implemented using the Izhikevich model. The parameters that have been used are adapted from [22] and correspond to regular spiking excitatory and fast spiking inhibitory neuronal cells. Also, heterogeneity among the neuronal populations has been achieved by allowing a stochastic variation in some of the parameters of the neurons. Again, this method was taken from the aforementioned paper.

The network is divided into specialized modules. Each of these modules inherits a specific structure which depends on the category that this module falls into. The categories that have been used are the followings:

- Workspace modules
- Sensory modules
- Motor modules
- Behavioural modules
- Action selection modules

In the rest of this chapter a description for each of these categories is provided along with lists and schematics for each module that is of particular interest.

Notation

The majority of the figures in this chapter concern the aforesaid schematics and thus follow a general structure. Mainly they consist of boxes of different sizes that represent groups of neurons that have the same characteristics and connectivity. These boxes are also connected with different lines that represent the different weighted connections (synapses) between neurons of the underlying groups. In particular, four types of lines have been used (see Figure 3.1.a) First, the lines that end with arrows represent excitatory connections, i.e. connections with a positive weight. On the other hand, the lines that end with circles represent inhibitory connections, i.e. connections with a negative weight. Finally, the dotted lines represent topographical excitatory or inhibitory connections.

A topographical connection between two groups of neurons of the same size is defined as an ordered projection of the neurons from the first group into the second (see Figure 3.1.b and c). This type of connectivity has been used in cases that the activation pattern of some neurons has to be preserved, either unchanged or modified.

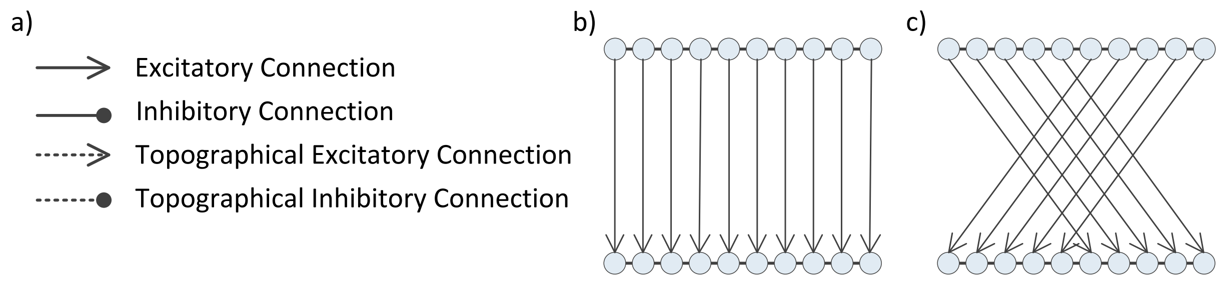

a)
Excitatory Connection
Inhibitory Connection
Topographical Excitatory Connection
Topographical Inhibitory Connection
b)
c)

Figure 3.1: a) Notation followed by figures in this chapter b) Topographical excitatory connections that don't change the activation pattern of the neurons c) The activation pattern is now changed.

3.1 Neuronal Global Workspace

The global workspace provides the system with a working memory where all the different kinds of active modules can broadcast information (i.e. spiking patterns) that all other modules can access at any time. Also, it assists the process of competition and cooperation between behavioural modules and prevents the simultaneous control of motor output by two modules. In this design, the global workspace consists of four identical layers, named workspace modules, that are connected to each other in a chain manner. The connections are both excitatory and inhibitory. The excitatory connections are topographic and spread and maintain the patterns of activation within the workspace, forming a type of working memory. On the other hand, the inhibitory connections smoothly diminish the effect of the same patterns. As a result, they determine the duration of the working memory. The architecture of the workspace is heavily based on the workspace model in [32].

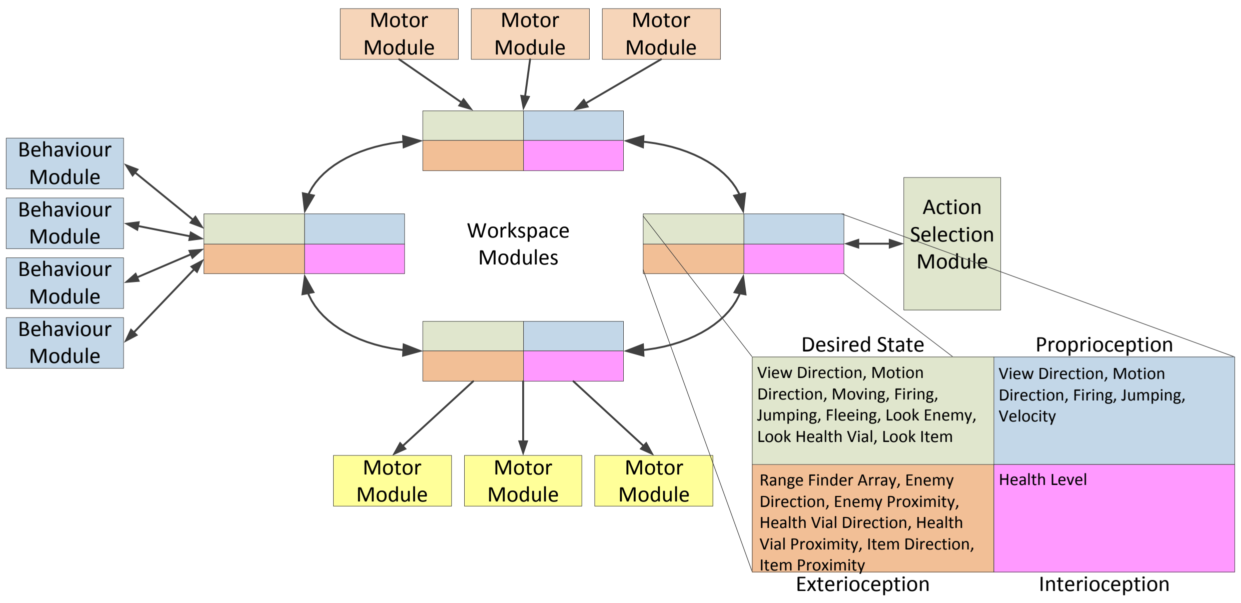

Motor Module
Motor Module
Motor Module
Behaviour Module
Behaviour Module
Behaviour Module
Behaviour Module
Workspace Modules
Action Selection Module
Desired State
Proprioception
View Direction, Motion Direction, Moving, Firing, Jumping, Fleeing, Look Enemy, Look Health Vial, Look Item
View Direction, Motion Direction, Firing, Jumping, Velocity
Range Finder Array, Enemy Direction, Enemy Proximity, Health Vial Direction, Health Vial Proximity, Item Direction, Item Proximity
Health Level
Exterioception
Interioception
Motor Module
Motor Module
Motor Module

Figure 3.2: Global neuronal workspace architecture

Each workspace node is connected and related to a different kind of cortical module (see Figure 3.2. As shown in this figure, the sensory modules of the system are all connected and try to get access to one specific workspace node while another workspace node is connected to the motor modules. Also, the behavioural modules are connected to a different node. Finally, the last node is dedicated only to one module, the action selection module.

Workspace Module

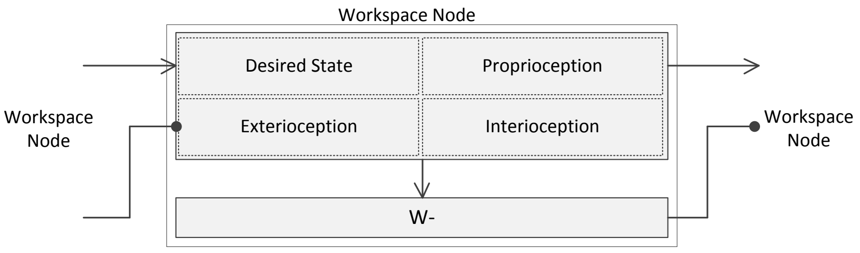


Figure 3.3: Workspace Node Structure

A workspace module consists of two neuronal pools, one excitatory and one inhibitory. As indicated in [22], the ratio of excitatory to inhibitory neurons is set to be 4 to 1, while the inhibitory synaptic connections are significantly stronger. The excitatory pool is divided into four areas that represent different types of globally accessible information. Each area can be further broken down into groups of neurons that correspond to different pieces of data.

The first area concerns the desired state of the body of the avatar while the second area represents the proprioceptive sensory data, i.e. the incoming information about the real state of the avatar's body. The third type represents the exteroceptive sensory data, i.e. the information about the visible part of the environment. Finally, the fourth type represents the internal state of the avatar and in this case includes only the level of its health condition. Table 3.1 shows the semantic contents of each of the above four areas along with the number of neurons that are allocated for each datum in the workspace and the types of values that they represent. Finally, the data that are extracted or sent back to the UT2004 environment, in each case of the sensory or motor modules, are also included to the same table.

3.2 Neural Coding

Each of the above datum in a workspace node represents either a binary, scalar or vector values and has been coded using either a rate coding method for the first two cases or a population coding methods for the latter. More specifically, binary or scalar values such as jumping, enemy proximity and health level are encoded/decoded based on the firing rate of 50 equally dedicated neurons which is set to be proportional to the intensity of the corresponding stimulus. Furthermore, egocentric one-dimensional neuron population vectors are used to represent spatial values in the avatar's environment, such as the direction of an enemy, the desired direction of movement or the incoming range finder sensor's data, always with respect to the avatar's point of view. In these vectors, the firing rate of each neuron represents a direction around the avatar which is relative to the current point of view.

To understand what is meant here, Figure 3.4 illustrates a possible situation and some corresponding neural encodings. In this situation, the avatar has noticed the presence of a wall on its left and as a result, the "navigation" behavioural module takes control of the desired motion direction. Distances of different angles between the wall and the avatar are captured from the range finder sensor and encoded using an egocentric one-dimensional population code. Then, different internal layers of the system produce a specific pattern in the firing activity of the output layer of the motor module that concerns the desired direction of motion. This activity is then decoded into a number that represents a direction in a 360 degree view around the bot with respect to the avatar's current point of view.

	Value	Type	Neurons	GameBots2004 data used
Desired State	View Direction	Vector	108	ROTATE command
	Motion Direction	Vector	108	MOVE command
	Moving	Scalar	50	MOVE command
	Firing	Scalar	50	SHOOT command
	Jumping	Scalar	50	JUMP command
	Fleeing	Scalar	50	ROTATE command
	Look Enemy	Scalar	50	ROTATE command
	Look Health Vial	Scalar	50	ROTATE command
	Look Item	Scalar	50	ROTATE command
Proprioception	View Direction	Vector	108	self rotation
	Motion Direction	Vector	108	self velocity
	Firing	Scalar	50	self shooting condition
	Jumping	Scalar	50	self jumping condition
	Velocity	Scalar	50	self velocity
Exterioception	Range Finder	Vector	108	19 cast rays
	Enemy Direction	Vector	108	player position
	Enemy Proximity	Scalar	50	player position
	Health Vial Direction	Vector	108	item position, item id
	Health Vial Proximity	Scalar	50	item position, item id
	Item Direction	Vector	108	item position
	Item Proximity	Scalar	50	item position
Interioception	Health Level	Scalar	50	self health level
	Total neurons		1564	

Table 3.1: The contents of a workspace node

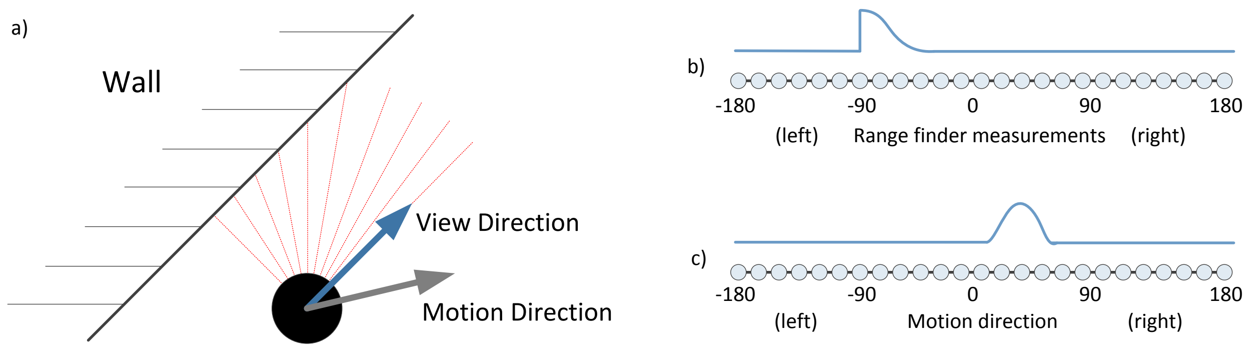


Figure 3.4: a) Representation of the avatar and the surrounding environment. b) Firing activity of the input layer of the "range finder" sensory module. c) Firing activity of the output layer of the "motion direction" motor module.

3.3 Sensorimotor Control

This system interfaces with UT2004 via a bot connection. Bot connections differ from real player (human) connections mainly on the available information that UT2004 broadcasts in each case. Although the goal of the bots is to interact with the same 3D environment like humans, they do not have access to the rich visual information that a human player has. On the other hand, bots have access to any numerical data that the server can provide such as the coordinates of a visible object or enemy or their distance from a wall.

More specifically, bots have direct access to the locations, directions and velocities of all the "visible" objects or avatars in the environment including themselves. Also, in order to have a better understanding of the surrounding environment, bots are also equipped with the ability to cast rays in any direction and receive the coordinates of the closest intersections with objects of the environment.

Furthermore, bots have access to boolean values that define different states of the avatar, such as jumping, falling or shooting as well as scalar values that describe the interioceptic state of the avatar such as level of health, armour and adrenaline.

Sensory Modules

Most of the above available sensory data are used as inputs to the system through specialized modules. These modules are injected with the relevant stimulus and compete to win access to the workspace. As shown in the Figure 3.5, there is a general structure followed by all sensory modules. However, the final architecture of each module depends on the type of data that they handle as well as any potential necessary internal processing of the incoming information.

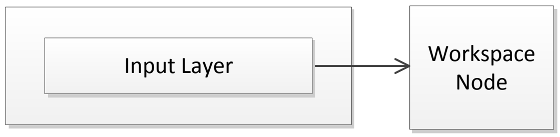


Figure 3.5: General architecture of a sensory module.

The types of data that have been used are binary, scalar or vector data. Also, in the current version of the system the majority of the sensory modules do not need internal processing of the incoming data. Modules that handle binary data are identical with the modules that handle scalars. Both consist of a single layer of 50 neurons and are connected to a workspace node. Modules that handle vectors also consist of a single layer but in this case the number of its neurons is 108.

The final data used by the system are the contents of the three areas of the workspace that represent proprioceptive, exteroceptive and interoceptive sensing and can be found in Table 3.1. Numerically, the scalar and binary values of this table are represented by a number from 0 to 1 and the vector values by an angle from -180 degrees to 180 degrees.

Before they are fed into the network, some of the incoming information needs to be pre-processed. The information about the closest visible item, closest health vial or the closest enemy is converted into a corresponding proximity scalar value and a vector that shows the direction of this item/vial/enemy with respect to the avatar's point of view. Furthermore, the system receives the intersection points of 19 predefined rays across the visible area of the avatar. These points are converted into distances of a simulated range finder sensor and fed into the network as a neuron population vector as shown in the Figure 3.4.

The results of the above conversions as well as the rest of incoming pieces of information are transformed into a mean firing rate λ, used to deliver spikes to the corresponding input layers with respect to the Poisson distribution of rate λ.

Motor Modules

Bots are also allowed to have a direct and very accurate control of the available motors by sending relevant commands. For instance they can send commands to define the next point they plan to move or the exact location of the object they plan to shoot.

In the present architecture, each motor module is governed by a single desired state and corresponds to a different command or short algorithm. Similarly to the sensory modules, these modules follow a general and very simple structure with a single output layer and connections to a workspace node (see Figure 3.6. Again, the size of the output layer depends on the type of the piece of information (i.e. desire) that the module is designed to handle.

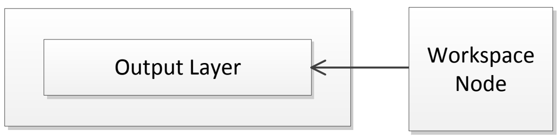


Figure 3.6: General architecture of a motor module.

At this point, nine motor modules are being used and correspond to the desired state area of the workspace. First, the desired "motion direction" and "view direction" motor modules as well as the "moving" module, control the MOVE and ROTATE commands that are sent to the avatar in every iteration. The two angles that are needed for these commands are extracted from the corresponding population vectors and are based on the neurons with the highest firing rate. Also, the MOVE command is sent only in case that the overall firing rate of the output layer of the "moving" module exceeds a threshold. Moreover, the same mechanism determines if the SHOOT and JUMP commands will be sent based on the firing rates of the"firing" or "jumping" modules.

Finally, the four remaining modules, are only used to set the value of a corresponding flag in order to print the desires of the avatar in the graphical interface and indeed in real time. However, in an alternative operation method of the system which will be discussed in Chapter 4, these values are used to correct the motion of the avatar.

3.4 Behaviours

A hypothetical neural system that includes the types of modules that have been described so far could sense, keep an internal representation of its understanding of the world and interact with the world. However it would not exhibit any willingness to respond to a stimulus because of the absence of behaviours. For this reason, one more essential category of modules was added to the architecture that defines the different responses of the system to various stimuli.

Like the previous cases, all the behavioural modules inherit a general structure (see Figure 3.7). An input layer of excitatory neurons is fully topographically connected with a workspace node in such a way that all the spiking patterns of the workspace node also pass to this input layer. Furthermore, it has internal connections with the salience layer and the internal layer. The salience layer, which consists of 200 excitatory neurons, represents the quality by which this behavioural module stands out relative to its competitors and thus assists the action selection mechanism of the system. It is also connected to the workspace node, and excites some neurons of the desired state area. Finally, the internal layer A can include both excitatory and inhibitory sub-layers depending on the case. Also, in some cases A can send spikes back to the workspace node.

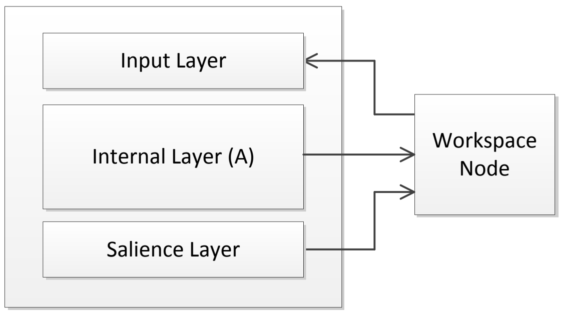


Figure 3.7: General architecture of a behaviour module.

The following behavioural modules are presented in the same order as the chronological order they were designed and tested using the implemented simulation software that will be discussed in the next chapter.

Moving Module

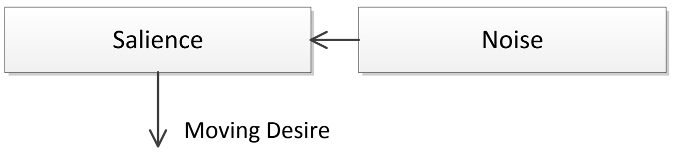


Figure 3.8: Architecture of the "moving" behavioural module.

This first behavioural module determines whether the avatar will move or not. However, it cannot be used alone since it could only generate a desire to move without providing any direction for the movement. Although this module constitutes the only exception among the behavioural modules because it does not follow their general structure, its function is fairly straightforward. A noise layer that always slightly excited, sends spikes to the salience layer which activates the moving desire. Because of the faint activation of the noise layer, the salience of the moving module is never very strong. Hence, a competitor module could very easily block its access to the workspace i.e. make the avatar stop moving.

Navigation Module

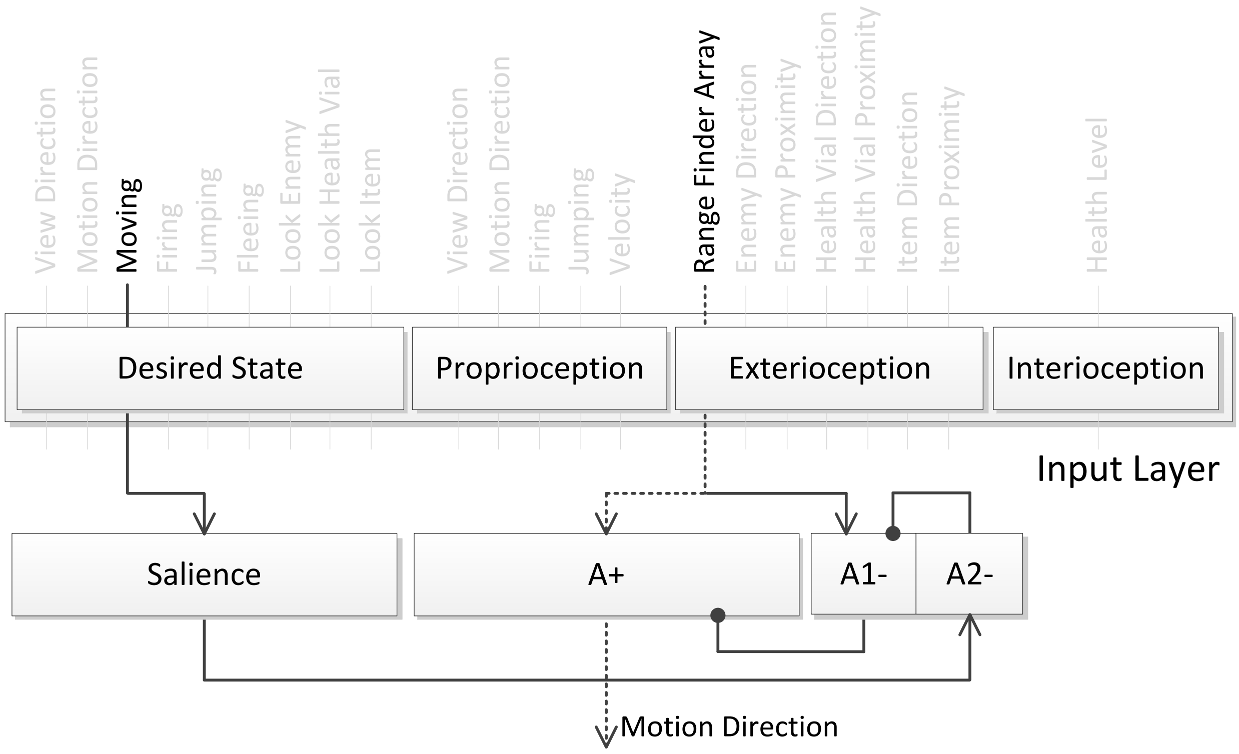


Figure 3.9: Architecture of the "navigation" behavioural module.

This module enables the avatar to walk alongside the walls and avoid collisions. In particular, when this module has access to the workspace, it controls the system's desired direction of motion. Together with the moving module it forms a coalition of processes that are able to win access and broadcast their information to the workspace at the same time.

In the navigation module, the neurons of a workspace node that represent the range finder measurements are topographically connected to the excitatory layer A+ which passes the same pattern to the desired motion direction. Hence, when this layer is active, the avatar is trying to move in the direction where the most empty space has been observed. However, the same neurons of the workspace also excite the inhibitory layer A1- which then inhibits A+. Thus, in a situation where the salience layer is not active, A+ is not able to send spikes back to the workspace node.

Furthermore, the salience layer gets activated when there is enough activation in moving desire. In this case, it sends spikes to the inhibitory layer A2- which then inhibits A1-. As a result, A1- becomes unable to stop A+ from sending the desired motion direction back to the workspace, and the entire module is considered active.

Exploration Module

Using only the above two modules, the system is able to move towards the view direction of the avatar. However, there is still a lack of a mechanism that controls this direction. The most essential situation in which the view direction needs to be controlled is when the avatar is placed in an environment without any observable payers of other interesting items. In this case, the avatar should exhibit an exploratory behaviour. In this implementation, this is carried out by the module in Figure 3.10.

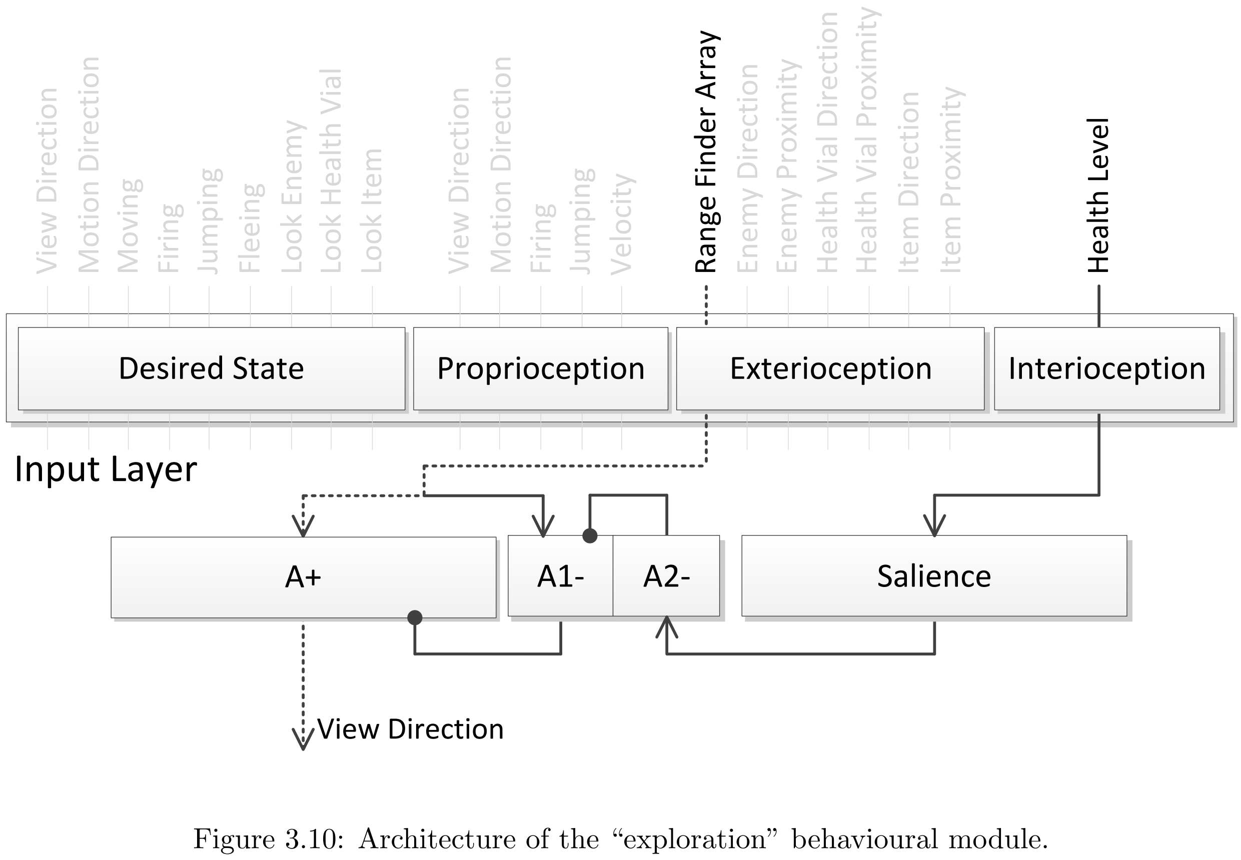


Figure 3.10: Architecture of the "exploration" behavioural module.

The architecture of the exploration module is fairly similar to the navigation module. The main difference is that when this module is active, it takes the control of the view direction of the avatar, rather than controlling the direction of motion. Also, the activation of this salience layer is proportional to the activation of the neurons that represent the level of health. The reason for this is to avoid exhibiting exploratory behaviours in cases where the health condition of the avatar is not good. Furthermore, since the control of the view direction is very coveted among the behavioural modules, the level of activation of this salience layer plays a more decisive role than the previous case.

The three modules, that have been described so far, can be considered as one of the simplest examples of a coalition of modules that cooperate to win access to the workspace together since they can be active simultaneously and the navigator module depends on the moving desire of the system. Hence, if the system is confident that it could win a potential combat, i.e. the health condition of the avatar is good, and there are no interesting objects or players visible then this salience level gets activated and the system becomes more "exploratory". In other words, it is able to navigate and explore any given environment following the closer walls and investigating large areas.

Firing Module

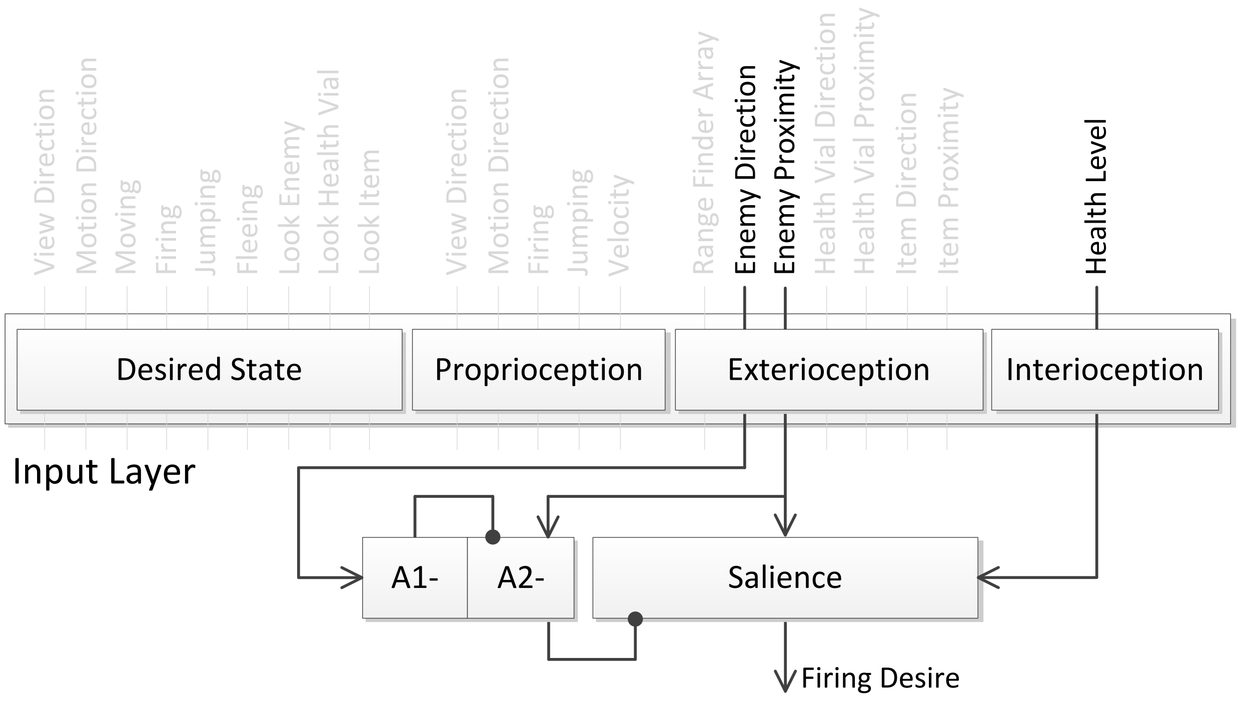


Figure 3.11: Architecture of the "firing" behavioural module.

The addition of this module to the system enables the avatar to defend itself when threatened by the presence of another bot or human player. Using a similar mechanism to the previous cases, the salience layer is activated only when the observed direction of the enemy is aligned with the current point of view. In order to achieve this, only the central neurons of the egocentric neuron population vector that represents the direction of the enemy are connected to the inhibitory layer A1-. Also, in this case, the activation of the salience layer is proportional to the enemy proximity and the level of health. Hence, if an enemy is close and the avatar has a sufficient health level, it will fire against this enemy.

Jumping Module

This module covers one more basic situation where the avatar needs to jump. According to its design, the system will send the JUMP command in two different cases or in any possible combination. First, if the system exhibits a desire to move, which is expressed by the corresponding neurons of the desired state area of the workspace, but it senses that it does not have horizontal velocity, the salience gets activated by the internal layer A+. This behaviour helps in circumstances where there is an obstacle in front of the avatar or the avatar is stuck because of another reason. Also,

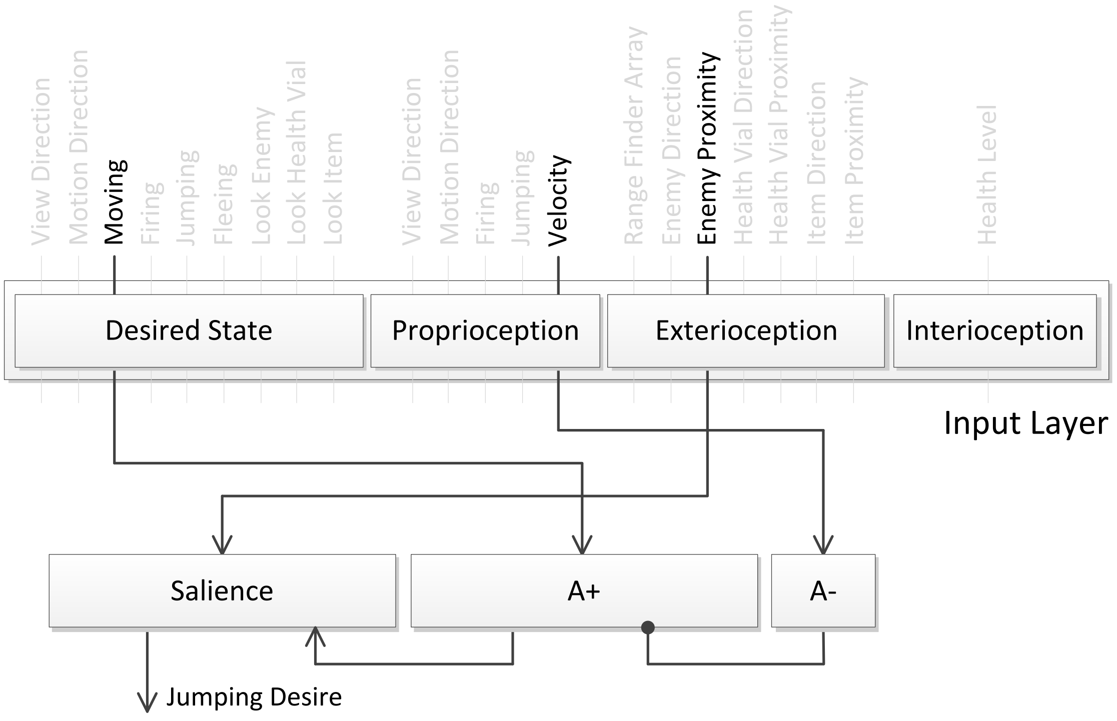


Figure 3.12: Architecture of the “jumping” behavioural module.

regardless of the avatar’s velocity, if an enemy has been observed and is approaching the avatar, it will also start jumping in order to avoid any possible firing.

Chasing Module

If the firing module is used, the bot is able to exhibit a defensive behaviour. However, it is very common for this category of video games, that human players and bots show aggressive behaviours. In fact, if a hypothetical player was only defending itself, they would never have an opportunity to win a match.

When the chasing module is active, it controls the desired direction of view and also activates the “look enemy” desire. The salience layer gets activated if an enemy is close and the health of the avatar is good. Then, the desired direction of view is aligned with the direction of this enemy.

The activation of this layer results a few different behaviours and depends on the moving desire and the firing desire of the system. For instance, when it is cooperating with the moving behavioural module, the result is to start chasing the enemy. Also, when the view direction is aligned with the direction of the enemy, it is possible that the firing module will start cooperating with the two other active modules and the avatar will start shooting at the enemy.

Fleeing Module

The controlling system is obliged to ensure the survival of the avatar. In some cases, this demands fleeing behaviour which in this implementation is performed by the fleeing behavioural module. According to its design, if an visible enemy is close to the avatar and the health level of the avatar is decreased, the avatar will start moving in the opposite direction of the enemy. To achieve this, the enemy direction is diametrically connected to the layer A+ which then sends back to the workspace node the direction that the avatar should point in order to flee from the enemy. On the other hand, however, the enemy direction also inhibits A+ through A1- in order to disable the broadcast of this vector in cases where the salience layer is not active. Furthermore, the activation of the salience layer is proportional to the enemy proximity, and inversely proportional to the level of health.

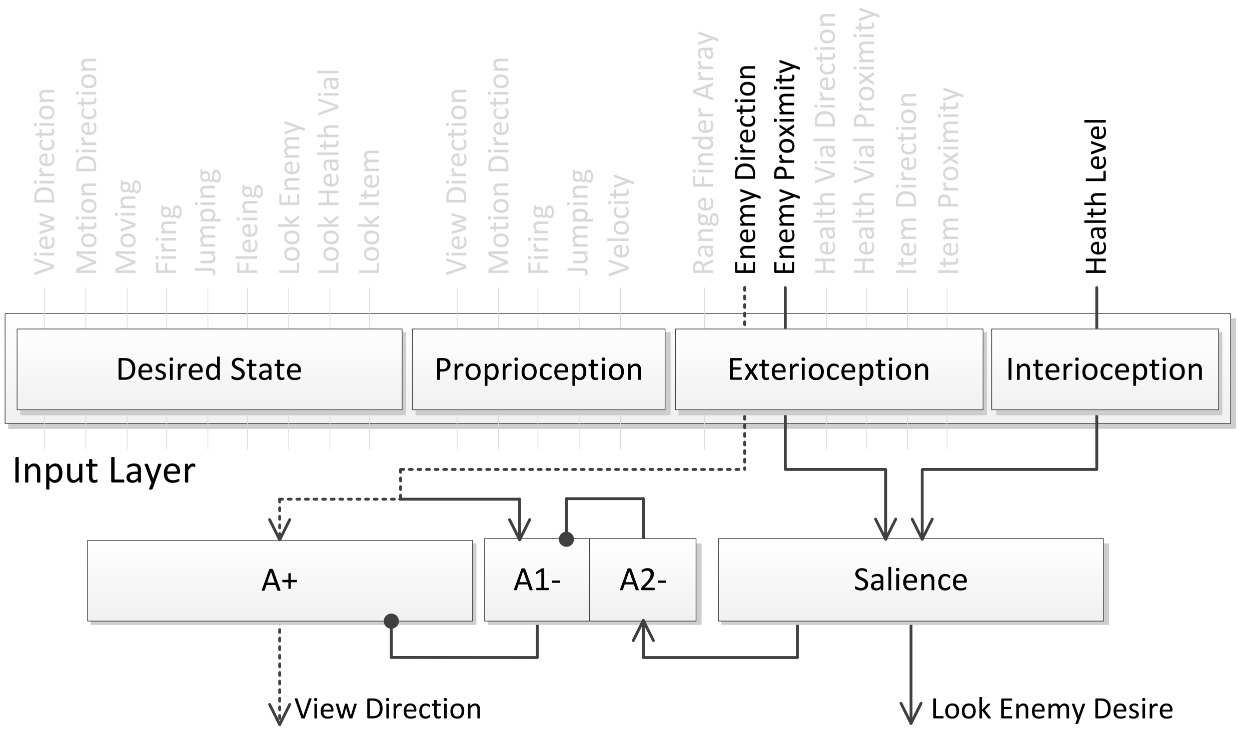


Figure 3.13: Architecture of the "chasing" behavioural module.

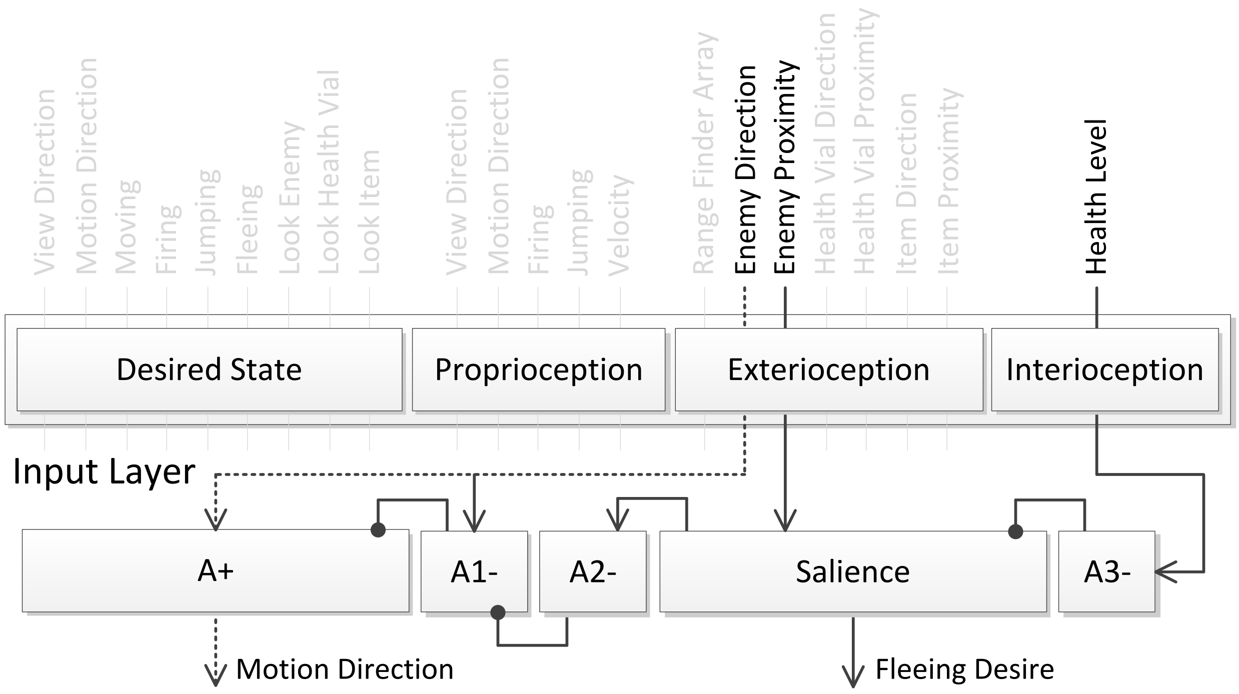


Figure 3.14: Architecture of the "fleeing" behavioural module.

Recuperating Module

In order to ensure the survival of the avatar, the system should also try to restore its health after an attack that caused damage to the avatar. Hence, if the health level is low and there is a visible health vial, this module will activate and will change the direction of view, making the avatar look at this health vial. If there are more than one visible health vials, only the closest one will be considered. This module usually cooperates with the fleeing and moving modules, which results in

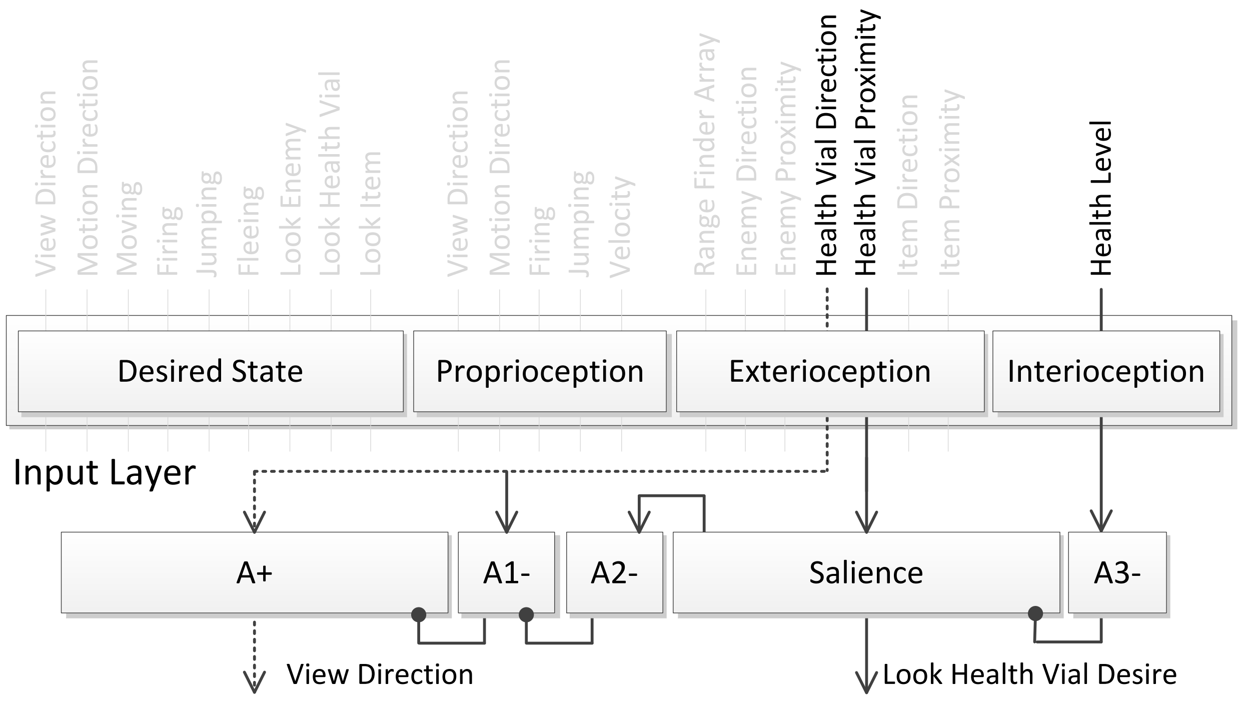


Figure 3.15: Architecture of the “recuparating” behavioural module.

a fleeing behaviour where the avatar is trying to avoid the attacking enemy and find health vials.

Look Item Module

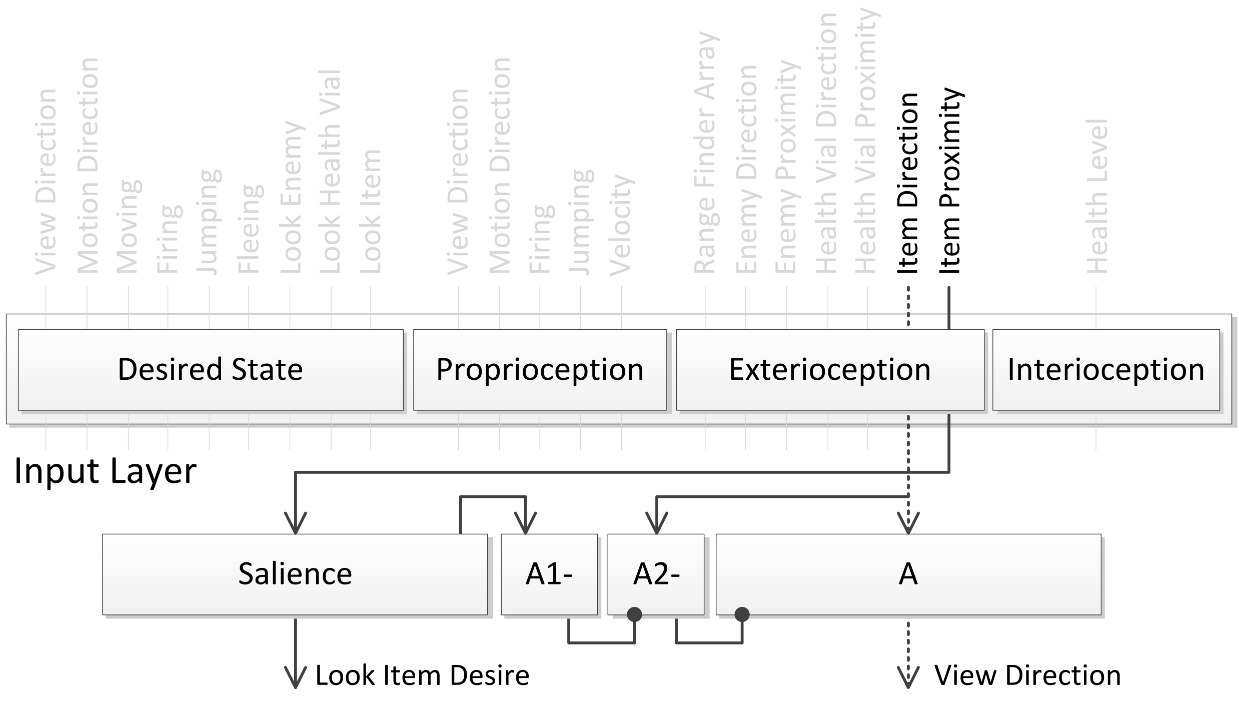


Figure 3.16: Architecture of the “look item” behavioural module.

This module controls the system’s desire to approach visible items. Its operation is quite similar to the previous module. When a visible item is noticed, the salience layer of this module becomes active and thus allows the A layer to control the view direction through a mechanism of two inhibitory layers (A1-, A2-) that work as a switch. The excitatory layer A sets the desired view direction to be aligned to the direction of the item. If this module as well as the moving and the navigator modules become active simultaneously, then they will result a behaviour where the avatar will approach and get the item-target.

U-turn Module

This last behavioural module was designed and added to the system after the basic implementation and testing. It is aimed to prevent situations where the avatar controlled by this system is surrounded by walls and cannot decide which direction is the best to turn. This usually means that the avatar is trapped and an obvious approach is to reverse the direction of motion.

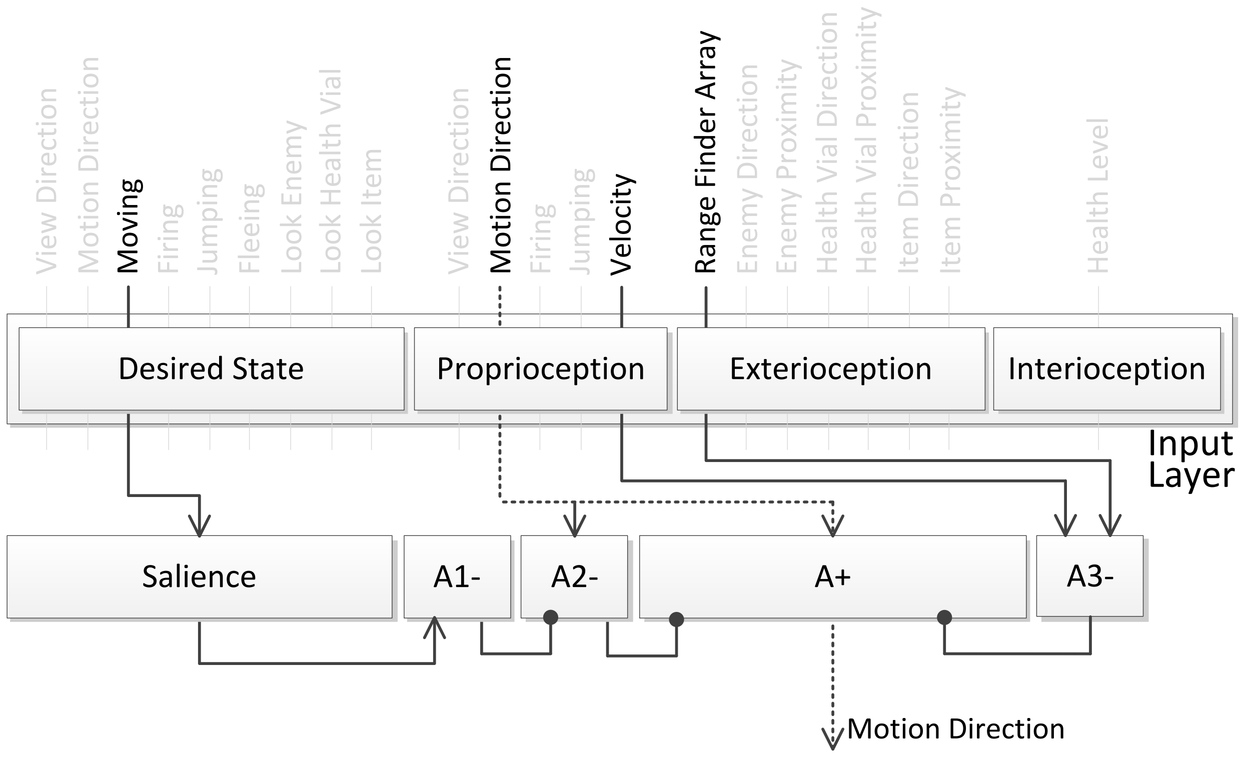


Figure 3.17: Architecture of the "U-Turn" behavioural module.

Therefore, A+ is diametrically connected with the motion direction and represents the direction that the avatar should turn in order to perform an 180 degree rotation. However, this layer is generally not active, because of the layer A2- which is also excited by the motion direction. The salience layer of this module is activated only when the agent feels the desire to move. In this case, the salience layer excites A1- which then inherits A2-. As a result, the A+ becomes active and starts broadcasting the altered motion direction. The layers A1- and A2- form the same "switch" mechanism that has been used in some of the previous modules. Finally, the feeling of high velocity or the observation of an open area with the range finder sensor excite the inhibitory layer A3-, which then inhibits the salience and A+ layers. Hence, the the module cannot broadcast information if the avatar is moving or not surrounded by walls.

A beneficial addition to this module would be the activation of the salience layer when the avatar is being attacked but the enemy proximity does not exhibit any activity. This generally means that another avatar is shooting at this avatar but they are not visible from this point of view. The reason that this improvement has not already be implemented is that it assumes the incorporation of the concept of pain which has also not been implemented at this point.

3.5 Action Selection

This special module provides rules for selection between the different desires that originate from the salience layers of the behavioural modules. Although it was not included in the initial design, it has been proven essential for the smooth conduction of the competition between some highly competitive behavioural modules. In particular, it was used to assist the competition between modules that concern desires of binary nature such as moving, firing or jumping. In a future version of the present system, this module could be integrated with the workspace modules.

It essentially consists of an excitatory input layer and an inhibitory layer of the same size. The input layer has the same structure and connectivity as the behavioural input layer described above but it interacts only with the first part of a workspace node which includes information about the desired state of the system. The input layer is connected topographically to the inhibitory layer which is also connected and inhibits the first part of the same workspace node. The exact connections that has been used in the current implementation are shown in Figure 3.18.

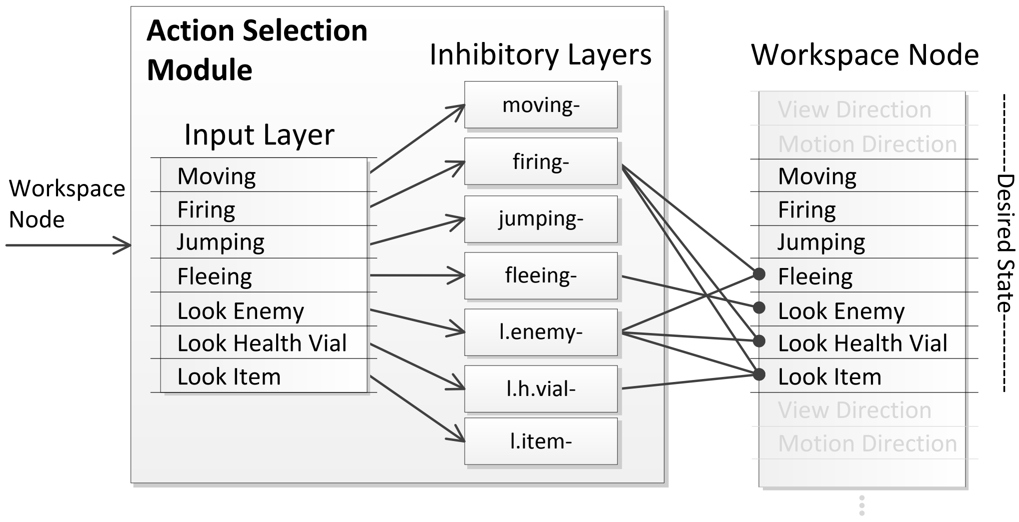


Figure 3.18: Architecture of the action selection module.

Chapter 4

Software Design and Implementation

The previous chapter provided an overview of the neural architecture that has been designed in order to control an avatar in a video game environment. This chapter describes the software that was developed to construct and simulate this neural network, enable it to interface with the UT2004 environment and test its performance. It also provides implementation details that have been considered important either because they play a decisive role in the operation of the final system or because they are related to technical difficulties that were overcome. The structure of the chapter was chosen to follow a chronological order, so as to be easier for the reader to understand the reasons of the implementation choices.

4.1 GameBots Parser

The first step of the software design and implementation was the creation of a basic program that is able to interact with Unreal Tournament 2004 (UT2004) using the GameBots interface. This program essentially consisted of a parser that can interface with GameBots based on the described communication protocol of GameBots and a number of objects that form the internal representation of UT2004 environment. After the development and testing, this program was used as the foundation for the development of all the other parts of the system's software.

To begin with, the parser was programmed in C++ and is based only on the Qt cross-platform application framework and C++ standard library. As a result, it can be easily compiled and operate on all popular operating systems. Also, regular expressions were used to achieve a satisfactory performance during the analysis of the incoming messages.

The main loop of the parser consists of two phases, the message analysis and the state update. Firstly, the program receives a message from GameBots server and using the regular expressions analyses it, using different predefined steps according to the entity that the message refers to. Then, the outcome of this analysis passes to the attributes of the objects that concern the internal and mathematical representation of the world.

The core of this representation can be found in a central object of a class named World (see Figure 4.1). This object holds all the necessary information regarding the controlled avatar, the visible items, players and navigation points as well as the past intersection points that were found from the ray tracing function. Also, it stores some statistics that can be used for the evaluation of the bot's performance.

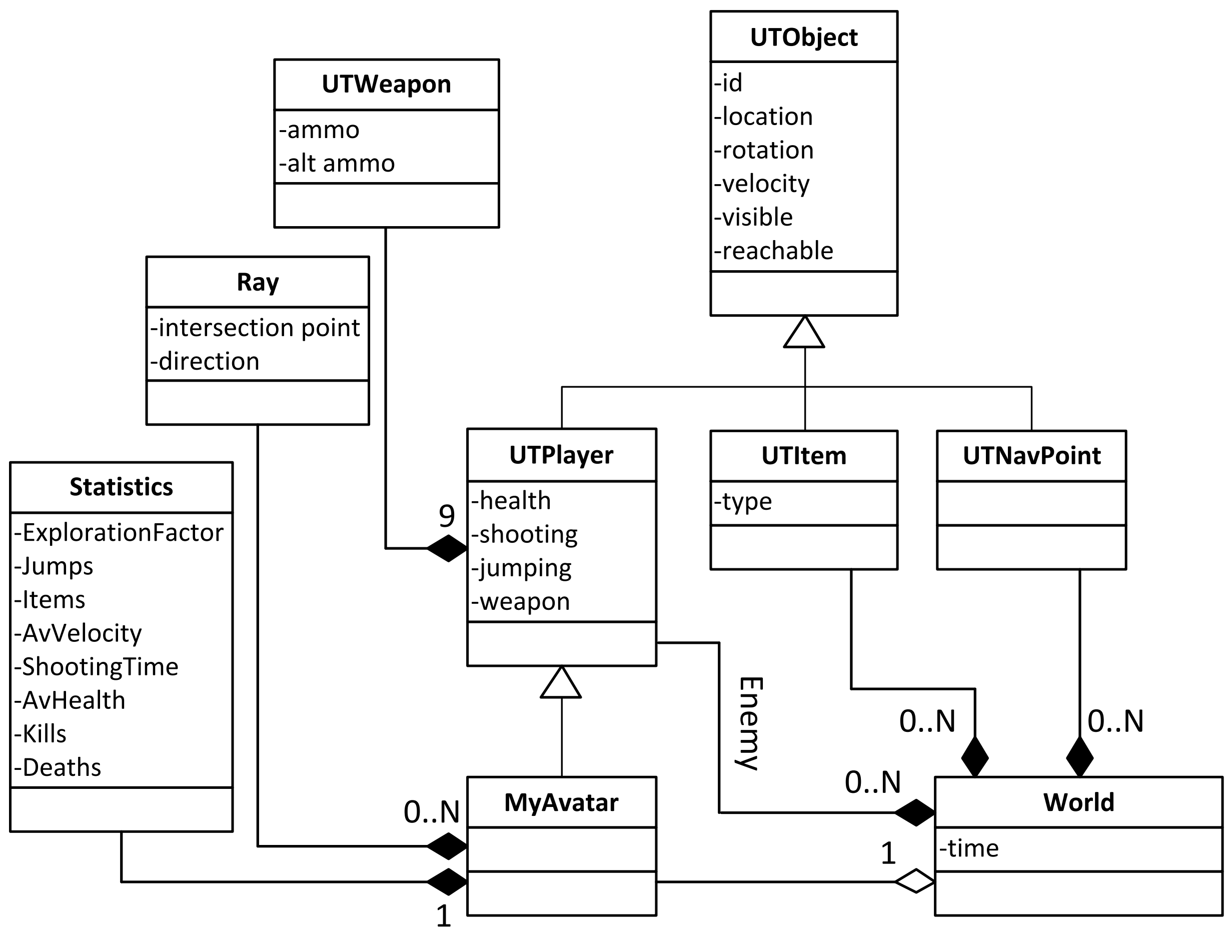


Figure 4.1: a UML class diagram that illustrates the classes that are related to the program's internal representation of the world.

The main reason for developing this parser rather than using an existing solution such as Pogamut 3 [18] was that none of the existing solutions were written in C++ which is the programming language that was planned to be used for this project. C++ language was chosen in order to avoid integration issues by minimizing the number of different technologies involved as well as because of its high performance. The final system was planned to be computationally very demanding with high hardware requirements that include continuous parallel processing using GPU. Hence, the performance of each module of the final software system was a very significant issue. Furthermore, NeMo simulator is also written in C++ and provides application programming interfaces (APIs) in C, C++ and matlab which, again, justifies the aforementioned choice.

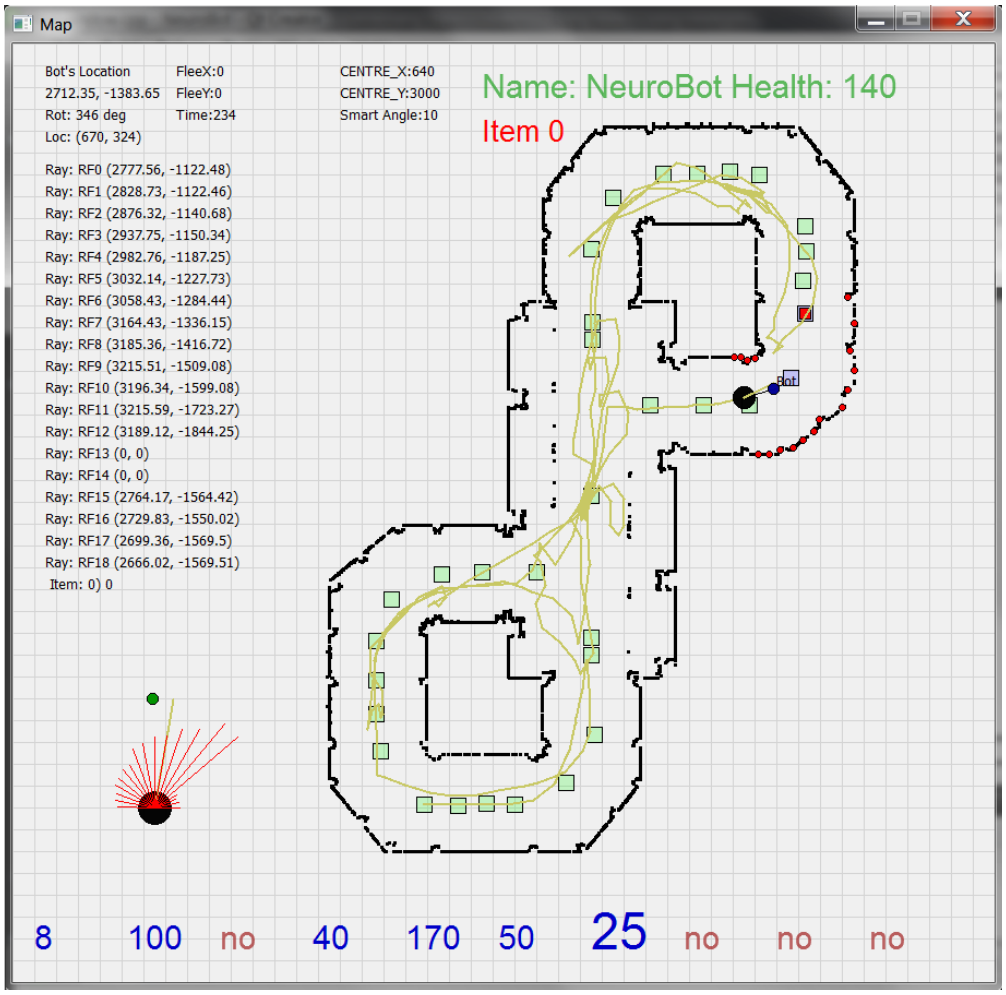
The performance of the final version of this program was tested and found to be satisfactory. In particular, it is able to operate in real time, without consuming too many system resources, even when the time period between two incoming messages is 30 times less ($0.01sec$) than the normal operation of GameBots server ($0.29sec$).

4.2 Graphical User Interface

The next step of the software development was the introduction of a graphical user interface (GUI) that allows the user to monitor and control the performance of the system.

Visualization

One of the basic components of the GUI is a visualization tool that illustrates a simple version of the UT2004's world as it is observed by the avatar in real time. In particular, the visualization that is produced consists of a plan that includes the observed points of the map's walls, the visible navigation points, items and players as well as the controlled avatar. Also, the visualization includes information about the health condition, the weapons that can be found in the avatar's inventory as well as a second model of the avatar that shows the distances with the surrounding walls and the best direction of movement according to these distances. Finally, the complete path that the avatar has followed from the beginning is also included. An example of this visualization can be found in Figure 4.2.



```
Bot's Location          FleeX:0        CENTRE_X:640
2712.35, -1383.65       FleeY:0        CENTRE_Y:3000
Rot: 346 deg            Time:234       Smart Angle:10
Loc: (670, 324)

Ray: RF0 (2777.56, -1122.48)
Ray: RF1 (2828.73, -1122.46)
Ray: RF2 (2876.32, -1140.68)
Ray: RF3 (2937.75, -1150.34)
Ray: RF4 (2982.76, -1187.25)
Ray: RF5 (3032.14, -1227.73)
Ray: RF6 (3058.43, -1284.44)
Ray: RF7 (3164.43, -1336.15)
Ray: RF8 (3185.36, -1416.72)
Ray: RF9 (3215.51, -1509.08)
Ray: RF10 (3196.34, -1599.08)
Ray: RF11 (3215.59, -1723.27)
Ray: RF12 (3189.12, -1844.25)
Ray: RF13 (0, 0)
Ray: RF14 (0, 0)
Ray: RF15 (2764.17, -1564.42)
Ray: RF16 (2729.83, -1550.02)
Ray: RF17 (2699.36, -1569.5)
Ray: RF18 (2666.02, -1569.51)
 Item: 0) 0
```

Name: NeuroBot Health: 140
Item 0

Figure 4.2: An screenshot of the Map window of the GUI.

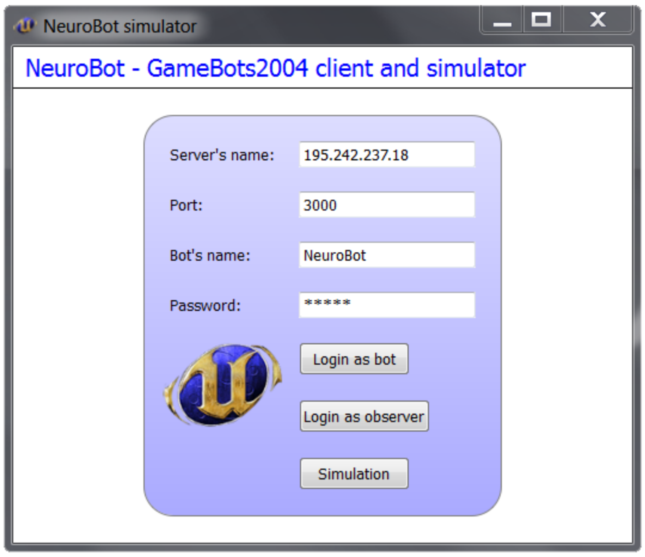
At this point, a division of the main program into different modes, according to the aim of the user, started to become necessary. In different situations, the user may either want to run a simulation in the game environment or connect to observe another player possibly to store some statistics. Hence, an option of choosing between two different modes was added to the program. The first mode was the bot connection mode and the second the observer mode.

Bot Connection Mode

In this mode the GUI is arranged in a way that both information regarding the controlled avatar and the system that is used to control it, are visible. In particular, all the features of the visualization tool that have been described above are enabled, and the window that illustrates the environment looks like Figure 4.2. However, the user can disable any of the illustrated features in order to speed up the processing of demanding algorithms.

When the application starts and this mode is chosen, a “hand-shaking” message arrives on the selected GameBots server which then constructs a new avatar and initiates a bot connection with this avatar. Also, a new thread is then created by the application with a view to execute the algorithms that govern this bot.

Finally, in this mode the user is able to take a full control of the avatar using the computer’s keyboard. This option is essential in situations that the avatar has been stuck or during experimentations where a human guidance is needed.


NeuroBot simulator

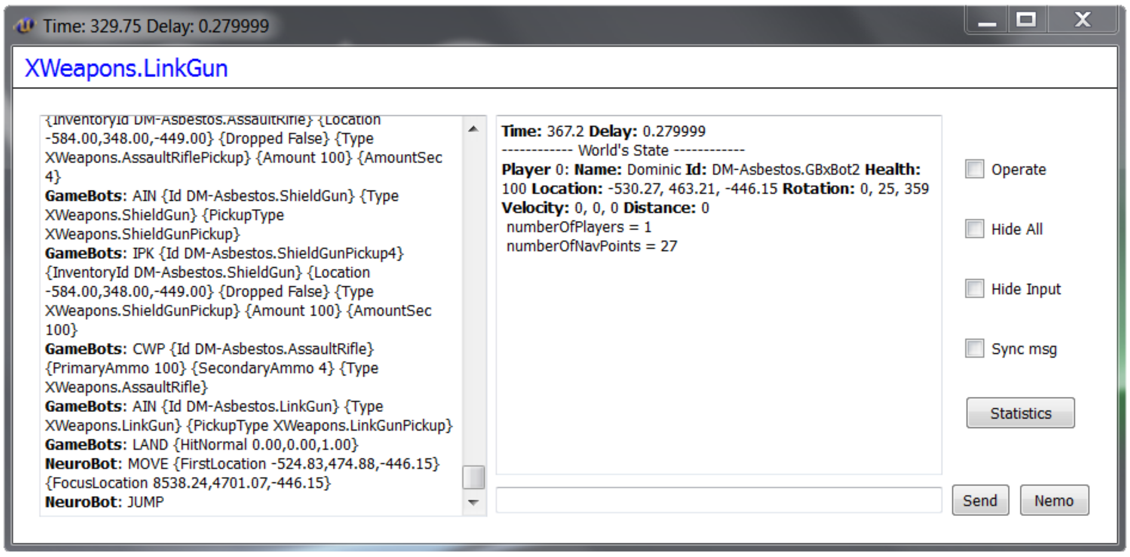
NeuroBot - GameBots2004 client and simulator

Server's name: 195.242.237.18

Port: 3000

Bot's name: NeuroBot

Password: *****


Time: 329.75 Delay: 0.279999

XWeapons.LinkGun

```
{InventoryId DM-Asbestos.AssaultRifle} {Location
-584.00,348.00,-449.00} {Dropped False} {Type
XWeapons.AssaultRiflePickup} {Amount 100} {AmountSec
4}
GameBots: AIN {Id DM-Asbestos.ShieldGun} {Type
XWeapons.ShieldGun} {PickupType
XWeapons.ShieldGunPickup}
GameBots: IPK {Id DM-Asbestos.ShieldGunPickup4}
{InventoryId DM-Asbestos.ShieldGun} {Location
-584.00,348.00,-449.00} {Dropped False} {Type
XWeapons.ShieldGunPickup} {Amount 100} {AmountSec
100}
GameBots: CWP {Id DM-Asbestos.AssaultRifle}
{PrimaryAmmo 100} {SecondaryAmmo 4} {Type
XWeapons.AssaultRifle}
GameBots: AIN {Id DM-Asbestos.LinkGun} {Type
XWeapons.LinkGun} {PickupType XWeapons.LinkGunPickup}
GameBots: LAND {HitNormal 0.00,0.00,1.00}
NeuroBot: MOVE {FirstLocation -524.83,474.88,-446.15}
{FocusLocation 8538.24,4701.07,-446.15}
NeuroBot: JUMP
```

```
Time: 367.2 Delay: 0.279999
------------ World's State ------------
Player 0: Name: Dominic Id: DM-Asbestos.GBxBot2 Health:
100 Location: -530.27, 463.21, -446.15 Rotation: 0, 25, 359
Velocity: 0, 0, 0 Distance: 0
 numberOfPlayers = 1
 numberOfNavPoints = 27
```

Figure 4.3: Up: A screenshot of the initial (mode selection) window of the GUI. Down: The window that shows the text information (synchronous/asynchronous messages and commands) that form the communication between the parser and the GameBots2004 server.

Observer Mode

In cases where this mode is chosen, the GUI displays only some of the aforementioned features. Since the observers cannot affect the state of the observed avatar in any way, features that concern ray tracing such as the illustration of the walls are not available here. On the other hand, since the application does not take up much of the computer’s resources, an update step can be performed significantly faster. As a result, the motion in the visualization is much smoother and the environment becomes ideal for observation and recording of statistics.

Following this vein, a new functionality was added to this mode that involved the recording all of the available forms of data and storing them in files. Hence, this mode was later used for the experimentation and statistical analysis that are described in section 5.1.

4.3 NeMo Integration

After the completion of the above, the development proceeded to the next stage which was the integration of the NeMo spiking neural network simulation environment [14]. This was an essential part of the implementation since it would make the system able to support algorithms based on spiking neural networks.

There were two main additions to the system. First, a new thread which can run independently of the main program was added. In this thread, one can initiate and run a simulation of spiking neurons using the C API of NeMo. This thread can also interact with the other windows of the system exchanging information. Also, taking advantage another feature of NeMo, the system became able to automatically detect whether the hardware on which it is running has a compatible graphics processing unit (GPU). If this is the case, the system is able to calculate the next state of the neural network using this GPU.

Furthermore, the second addition to the system was a new window that enables the user to visualize all the necessary pieces of information that come from the simulation. For this reason, two general classes of different animated graphs were developed. Using instantiations of these classes, the NeMo simulator window is able to illustrate the spiking patterns of each neuron of a network, their membrane potential as well as other measures, always with respect to the time. Figure 4.6 shows a screenshoot of the NeMo simulator window where nine diagrams were used to show the behaviour of different layers of neurons that control a simple simulated robot in a maze.

4.3.1 Encoding and Decoding

In order to make the neural network able to interact with the simulated environment, it is necessary to convert the scalar values that contain information into patterns of spikes that can be fed into the network and vice versa. As mentioned in Section 3.2, two methods have been used for neural representation of values, the rate coding and the neuron population coding methods.

The encoding of a scalar value using the rate coding method can be easily handled by NeMo which is able to provide external stimulus to the network, by forcing specific neurons to fire at a specific frequency. Also, the decoding of the same values only requires the calculation of the average firing rate of the corresponding neurons, with respect to time. This again can be considered fairly straightforward.

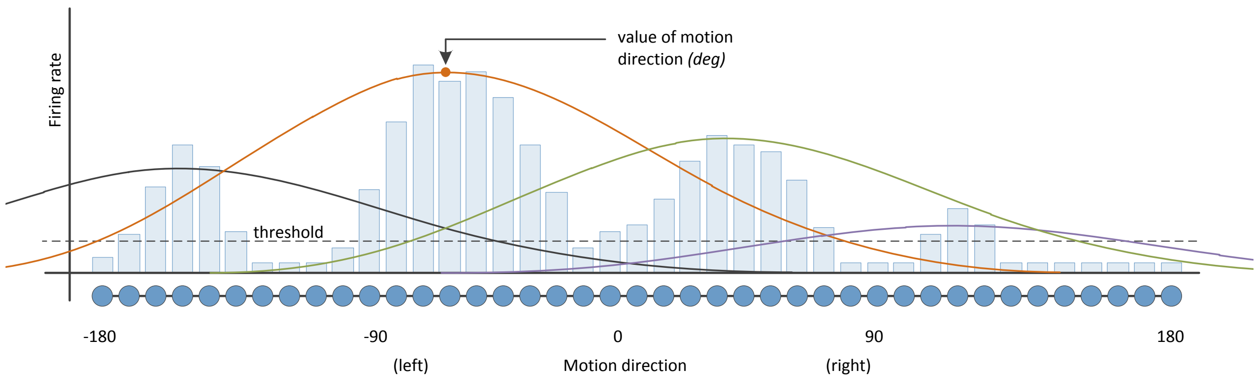


Figure 4.4: Example of decoding the value of the egocentric one-dimensional neuron population vector that represents the desired motion direction

On the other hand, the population coding method is more complicated and the decoding procedure accompanied by some issues that need further investigation. Although the encoding could be easily performed by stimulating only the neurons that correspond to the desired angle, this is not the case with the decoding. The reason is that all neurons of a motor layer are likely to have a positive firing rate. Hence, in order to find the value of a neuron population vector, it is necessary to find the centre of the area of that vector that exhibits the highest activation. This can be found using the following algorithm. In this algorithm, the neuron population vectors are considered as discrete functions $f(n)$ where $n = 1, 2, ..., k$, k is the number of neurons of this vector and $f(n)$ is equal to the firing rate of the neuron n.

1. Find all the areas of the population vector f where the firing rate of each adjacent neuron is above the threshold T.
2. For each such area i, create a neuron population vector f_i that inherits the firing rate of the neurons of this area but all the other firing rates are set to be 0.
3. For each f_i, calculate the polynomial p_i which results from a 2nd order polynomial regression.
4. For each i, calculate $\max_x p_i(x)$.
5. If I the area with the maximum value, return the angle a such that $p_I(a) = \max_x p_I(x)$.

The Figure 4.4 illustrates an example where the desired motion direction population vector exhibits strong activity in four different areas. Hence, four different polynomials are calculated that correspond to four different vectors. The second polynomial is the one with the highest value which can be found in $p_2(-63.7)$, and therefore $a = -63.7$ is the value of the corresponding angle is sent to the simulated robot or avatar.

4.3.2 Debugging Mode

As mentioned before, the thread and the window that run and visualize the NeMo simulations are independent of the other parts of the system. This means that active connections to GameBots servers are supported but not required, when running a simulation of a spiking neural network using the present software. Taking advantage of this feature, a third mode was added to the system which supports independent simulations of neuronal robot controllers in a simple environment. The internal representation of this environment has the same structure as the one developed for the representation of the UT2004. Therefore, the transformation of a controller that has been developed and tested on this simple environment, to a version that controls an avatar within UT2004, turns out to be fairly straightforward.

Two samples of networks that carry out particular functions were developed in the debugging mode:

Braitenberg's vehicle Simulation

The first sample simulation concerns a controller of a differentially driven robot that uses distance sensors to navigate in a world that includes a number of walls (see Figure 4.6). The architecture of this controller was based on the classical Braitenberg vehicle of type 3b [11] and it consisted of 6 neuronal layers that are shown in Figure 4.5. During this simulation, the two excitatory sensory layers are injected with the stimulus that comes from the two corresponding sensors. Then, each of the sensory layers excites the motor layer that corresponds to the opposite wheel of the robot. This motor layer represents how this wheel should slow down in order to avoid the detected wall.

Moreover, layers A1- and A2- form a winner-takes-all mechanism which allows only one motor at the time to be slowed down, through mutual inhibition. The reason that this is an important feature of this controller is that it allows the robot to make decisions about which direction to turn towards, and stick with its decision. This is essential in order to avoid collisions with walls in cases where the right and left sensory layers are stimulated equally.

The architecture of this controller was intended to be used for the "navigation behavioural module" described in section 3.4. In this approach, the avatar is considered as a differentially driven vehicle. After some initial tests, it was shown that the task of navigation could be performed well using this approach. However, the behaviour of the avatar, turned out to be less human-like than the current version of this behavioural module and thus it has been omitted from the final system.

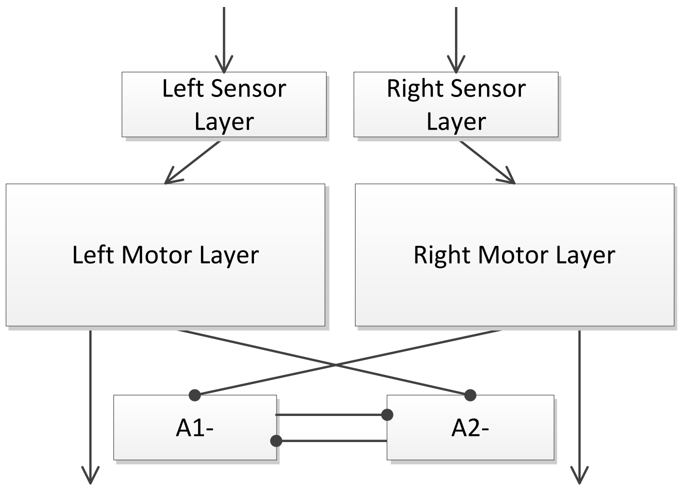


Figure 4.5: Architecture of the Braitenberg vehicle like neuronal controller.

Global Neuronal Workspace Simulator

This second sample simulation was the first attempt at the implementation of the neural architecture that has been described in Chapter 3. The main advantage of using this mode first, comes from its convenience for debugging. In this case, the input stimuli of the system can be completely controlled using keyboard shortcuts and the mouse. Also, the speed of the simulation can be decreased or even paused without this having an influence on the simulated environment since the latter also runs in the same thread. Therefore, the execution of the simulation can be less computationally demanding which allows more advanced visualizations to run at the same time.

For the above reasons, firstly, the neural architecture had been implemented in full depth, in this simple simulation environment. As a result, a number of design weaknesses were revealed and then resolved. However, at this point, the system's level of success it terms of its main goal could not be tested since this would require interaction with the world of UT2004.

4.4 NeuroBot Simulator

The final step for the development was the integration of the main controller that is described in Chapter 3 into the rest system in a way such that it would be able to control an avatar within the UT2004 environment. This was easy to achieve, since the global neuronal workspace simulator that is described above was constructed using the same principles and thus was completely compatible. The name that was chosen for the final system was "NeuroBot".

Again, after the first tests of the almost-completed system, different design issues were revealed and finally resolved. The remainder of this chapter provides a description of these issues along with the chosen solutions.

The most important issue that came out was that the level of accuracy when targeting an enemy was not good enough in order to reach that target. The reason for this was that the neuron population vectors that are used in the architecture to represent angles such as the view direction do not have enough neurons to achieve satisfactory resolution for successful targeting. On the other hand, an increase in this number of neurons would result a significantly larger network that would be much more difficult to be handled by conventional hardware.

Another issue could be observed in some cases where the avatar approaches and finally takes an item. This problem comes from the fact that the speed that the state of neural network is updated is significantly faster than the speed of the incoming messages that update the state of the world. Thus, some times the message that the item was taken is not delivered fast enough and the avatar keeps rotating around the location of this item believing that the latter is still there. The modification of the system in order to overcome this problem was simple. If an item is closer to the avatar than a threshold, then this item is not considered visible.

In the initial design of the architecture, it had been assumed that the world around the avatar is flat. Hence, all the directions were represented using one-dimensional neuron population vectors i.e. polar coordinates. Although the majority of the maps in UT2004 include several levels of different heights, this initial approach works in most cases. For instance, motion directions have to be of equal altitude since the avatar is not able to fly. Also, if an item is close to the avatar, it has to be at the same level. However, polar coordinates are not sufficient in the case of the direction of the approaching enemies or when the avatar should target an enemy in order to start shooting.

A final issue was the lack of a mechanism inside the neural architecture that enables the system to choose between different weapons and kinds of ammunition.

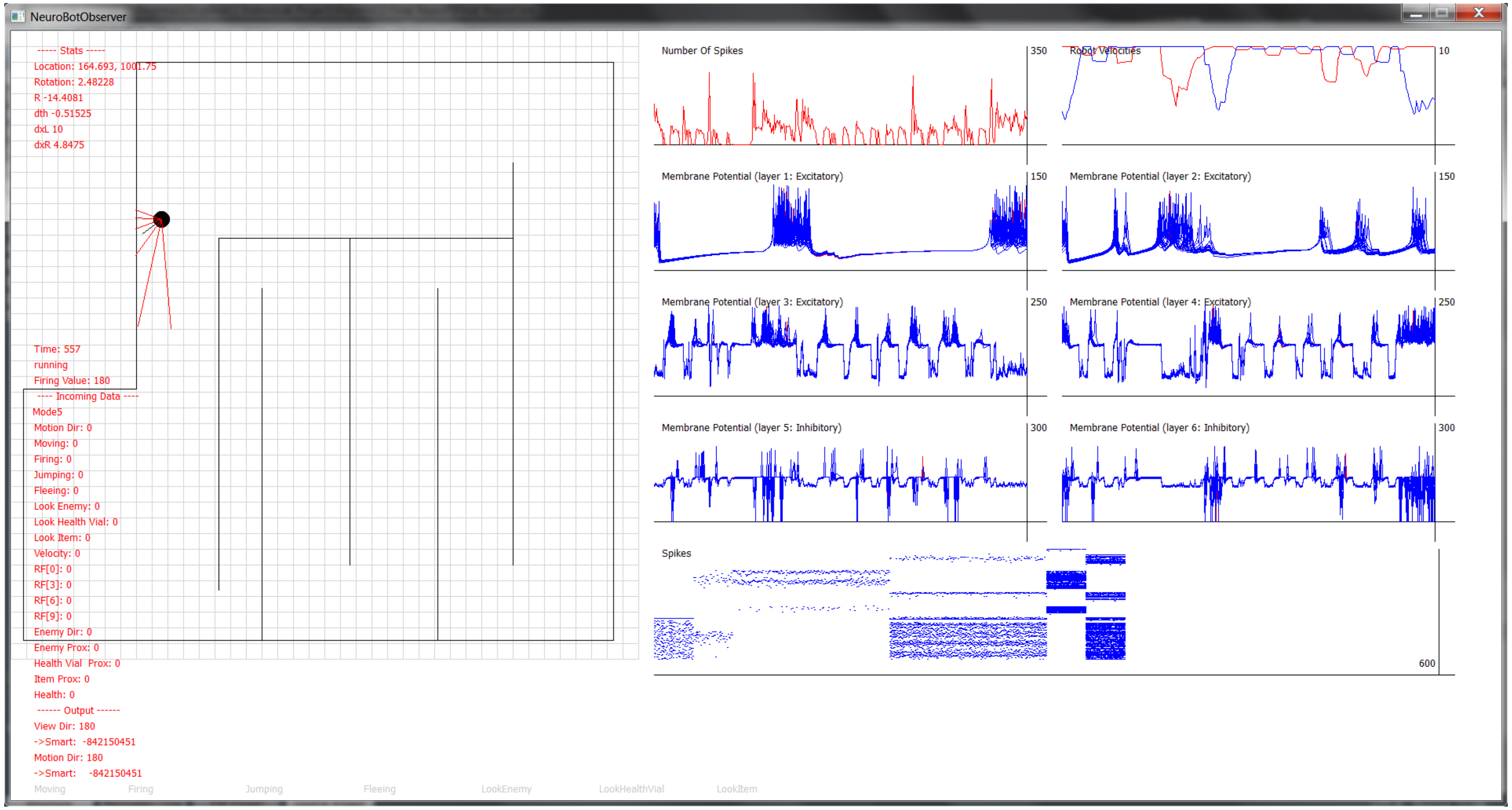
Error Correction System

In order to overcome the aforementioned issues and improve the performance of the controller in general, a helping subsystem was developed and named ECS which stands for "error correction system". If the final user wants, they are able to activate this subsystem that provides methods to correct the output errors and improve the overall performance. However, the downside of using this subsystem is that the operation of the avatar becomes less biological realistic, since the avatar is not entirely controlled by the neural network.

The issue of the level of accuracy when targeting an enemy was overcome by taking advantage of the numerical value of the direction of that enemy. When the ECS is activated and the view direction of the avatar is very close to the real direction of the enemy, then this direction is corrected automatically in order to point directly to that enemy.

A second improvement that comes with ECS concerns the 3D nature of the UT2004 maps which, as described above, sometimes leads to a need for changing the altitude of the view direction. The only case where this is necessary is when an enemy is approaching. Hence, in case that the players are not in the same level, a second angle is used to define the spherical coordinates of the enemy. This angle is calculated numerically by the ECS and used independently of the network in cases that the salience layer of the "look enemy" behavioural module is activated.

The last modification of the system concerns an algorithm that is used for weapon selection. This algorithm works also independently of the neural network and this is the reason why it is considered as a part of ECS. According to this algorithm, the weapons that have a greater index are considered better and have priority. Also, for each weapon this algorithm uses a predefined probability in order to choose the kind of ammunition that the avatar will use at each shoot. For instance, if the avatar is holding a "link gun", which is one of the weapons of UT2004 that its alternative ammo is considered to be more efficient than the primary ammo, then at each shoot there is a 40% probability of using the primary ammo and a 60% probability of using the alternative one.


NeuroBotObserver
Number Of Spikes
350
Robot Velocities
10
Membrane Potential (layer 1: Excitatory)
150
Membrane Potential (layer 2: Excitatory)
150
Membrane Potential (layer 3: Excitatory)
250
Membrane Potential (layer 4: Excitatory)
250
Membrane Potential (layer 5: Inhibitory)
300
Membrane Potential (layer 6: Inhibitory)
300
Spikes
600
Moving
Firing
Jumping
Fleeing
LookEnemy
LookHealthVial
LookItem

```
----- Stats -----
Location: 164.693, 1001.75
Rotation: 2.48228
R -14.4081
dth -0.51525
dxL 10
dxR 4.8475

Time: 557
running
Firing Value: 180
---- Incoming Data ----
Mode5
Motion Dir: 0
Moving: 0
Firing: 0
Jumping: 0
Fleeing: 0
Look Enemy: 0
Look Health Vial: 0
Look Item: 0
Velocity: 0
RF[0]: 0
RF[3]: 0
RF[6]: 0
RF[9]: 0
Enemy Dir: 0
Enemy Prox: 0
Health Vial  Prox: 0
Item Prox: 0
Health: 0
------ Output ------
View Dir: 180
->Smart:  -842150451
Motion Dir: 180
->Smart:    -842150451
```

Figure 4.6: Up: A screenshot of the graphical window that shows the output from NeMo as well as an instance of the Braitenberg's vehicle simulation. In this case, neural controller is guiding a simulated robot through a maze.

Chapter 5

Evaluation

The next step of this project after the implementation of the software system is to evaluate the performance of the system, i.e. to measure its ability to control an avatar exhibiting human-like behaviours within UT2004 environment. This was achieved by conducting two different evaluation tests. The first consisted of two experiments where this system (NeuroBot) played against three different skilled human players and a common bot. Both experiments lasted for 200 minutes and every 10 minutes the map changed randomly. For the first experiment the BotPrize server was used which was also available for other players to join, while in the second experiment NeuroBot was playing against one other player who was changing at each round. For this test, a number of different measures were defined and used in order to compare the behaviours of the aforementioned players during these experiments. The second test was the participation in the 2K BotPrize human-like bot competition at CIG 2011 in Seoul, South Korea where the performance of NeuroBot was tested against other approaches.

5.1 Statistical Analysis of behaviour

5.1.1 Measures

This section provides an overview of the different statistical measures that were used for the comparison of the bot's behaviour as well as a justification for each of these choices.

Exploration factor A GameBots map in UT2004 has predefined points that are spread throughout the accessible areas and placed in strategic positions in order to help simple software bots to navigate through the map without very complicated calculations. These points are called navigation points. A simple way to divide a map is to make N subsets where N is the number of navigation points on that map. Any point of the map then will belong to the set of the navigation point that is closer to this point.

The exploration factor that will be used is defined as the average (per minute) of the number of different Navigation Point Sets (NPSs) visited, divided by the number of changes of the current NPS, multiplied by the number of NPSs visited generally divided by the total number of NPSs in the map. The minimum value is 0 and the maximum is 1. This has been considered as an significant measure since it characterizes the level of exploration behaviour that a human player or a software bot exhibits.

$$Explorationfactor = \frac{1}{mins}\sum_{t=0}^{mins}\left(\frac{NPS_{visited}(t)}{NPS_{changes}(t)}\right)\times\frac{NPS_{visited}}{NPS_{total}} \tag{5.1}$$

If the observed player visits all the NPSs once, the exploration factor will be 1 which means that the avatar has followed the best possible path to explore the given map. Contrariwise, if the avatar alternates between two NPSs then the total number of visited NPSs will be 2 and the exploration factor will be ≈ 0 Also, if the player visits more NPSs but the most of the time alternates between two NPSs then the first part of the formula will reduce to 0 and thus the exploration factor will again be ≈ 0.

Average health A second measure that has been used and considered significant is the average level of the health of this player during a 10-minute round. This measure was used to indicate how important it is for the target player to heal themselves rather than participating in a fight. In other words this measure can indicate whether the player exhibits risk-taking behaviours.

Number of jumps This measure is simply defined as the total number of jumps of a player during a 10-minute round. The main reason of using this simple measure is because it is a common way that experienced human players use in order to distinguish bots from humans. Human players usually tend to jump more often than bots. This happens for a number of different reasons. First, in the majority of first person shooters when an avatar is jumping it can navigate faster. Also, when a player is jumping they are more likely to dodge incoming missiles or other kinds of incoming fire.

Average velocity Since the magnitude of the horizontal velocity of the avatars is constant while they are running, this measure can be considered as the easiest way to check how much time a player spent moving.

Shooting time This measure is again defined as the total time that a player spent for shooting, during a 10-minute round. This simple measure can indicate how accurate a player is in shooting. Also, it could be translated as the level of aggressiveness of a player.

Items taken The total number of items taken may indirectly indicate the intentions of the player during a round. If a player is very keen on participating in fights it would have less time to collect items than another player who does not give priority to specific tasks. Thus, this is an example of a very simple measure which can be linked to high level cognitive functions of a player.

Kills and deaths The reason of using this measure is quite obvious since it represents the ultimate comparison for the performance of each player.

Visualization of the followed path The final measure that has been used is a visualization of the path that this player has followed during a 10-minutes round. Although this is only a very informal way of comparing the humanness of players it is the simplest way of observing the level of randomness in the players motion.

5.1.2 5-player experiment

In this experiment NeuroBot played against 3 human players and another bot. The bot that was used is included in the standard distribution of UT2004 client and it is used to control the basic opponents of this game in other modes such as the single player mode. The server that was used for this experiment is the same server that supports the BotPrize competition which is programmed to change maps randomly every 10 minutes. During these 10-minute rounds, five observers were also connected to BotPrize server and they were linked to each of the five aforementioned players in order to record all the available extracted data. After 20 rounds, the average values of the measures described above were calculated and they are shown in Figure 5.1.

In some of the graphs of this figure, the close relation between the human players and NeuroBot is highly noticeable, in contrast to the case of the common bot. In particular, the first four players are more exploratory than the common bot and they spent quite less time seeking items. Also, NeuroBot jumped even more times that the human players, when the common bot jumped very few times. Finally, the three human players and the NeuroBot exhibited more risk-taking behaviours than the common bot, according to the average level of their health.

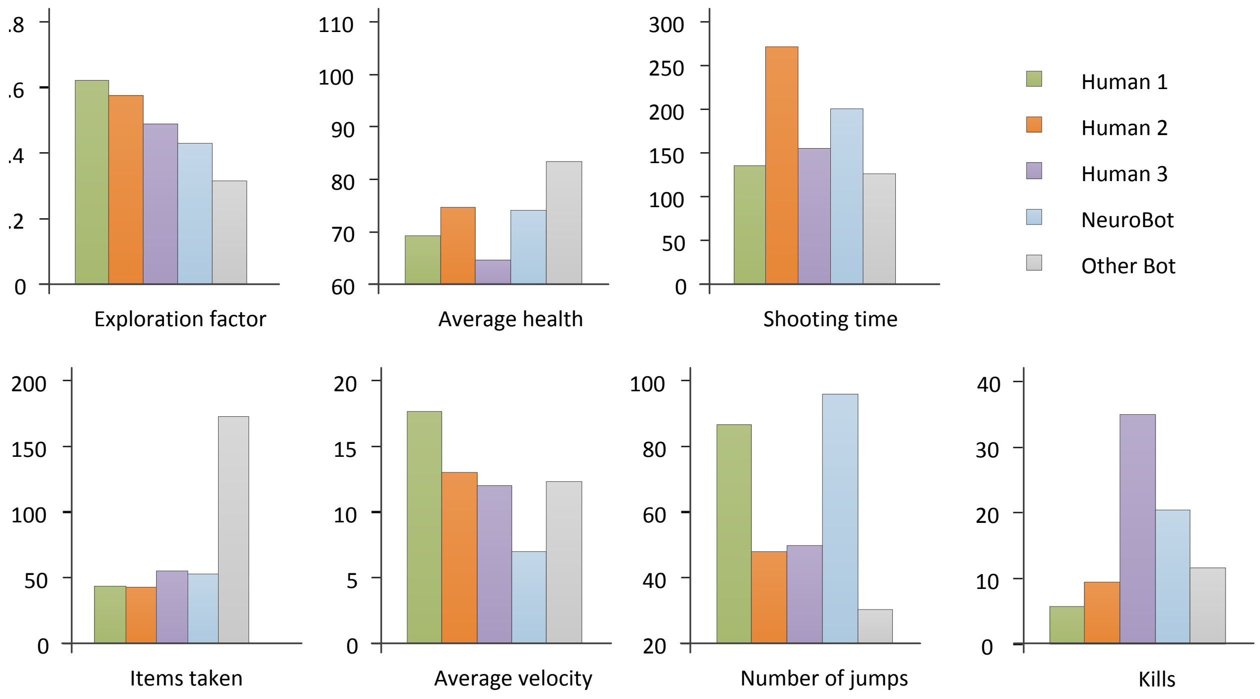


Figure 5.1: A graphical representation that shows the distribution of the recorded values of all players participating in the first experiment.

5.1.3 Player VS player experiment

In this second experiment the GameBots server allowed only two players connections to join at each round. The first player was always NeuroBot while the second player alternated between the other three human players and the common bot. Hence, in this configuration, the behaviours of each player in cases when there is one clear target could be observed. Again, after 20 rounds, the average values of the measures described above were calculated and they are shown in Figure 5.2.

According to this figure, one more time NeuroBot shown to have a closer relation to the human players than the other bot by exhibiting more exploratory behaviours but focusing less on seeking for items. Also, it spent more time shooting and trying to kill the opponent. However, in this case it jumped even more times than in the first experiment and exhibited even more risk-taking behaviours than the human players.

Furthermore, Figure 5.3 illustrates the paths that were followed by the NeuroBot and the common bot during a specific round of the second experiment. Here, the randomness of the first path is very noticeable in contrast to the second case where the navigation is highly dependent on the navigation points of the map (the small squares).

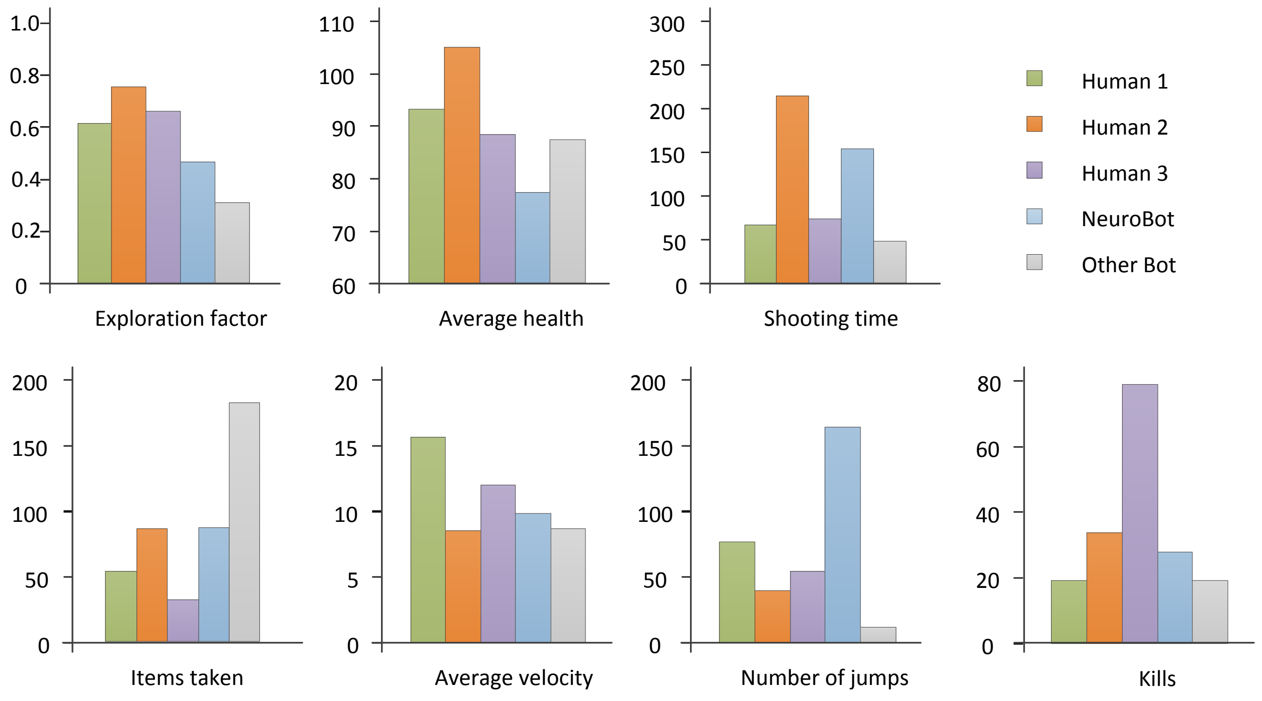


Figure 5.2: A graphical representation that shows the distribution of the recorded values of all players participating in the second experiment.

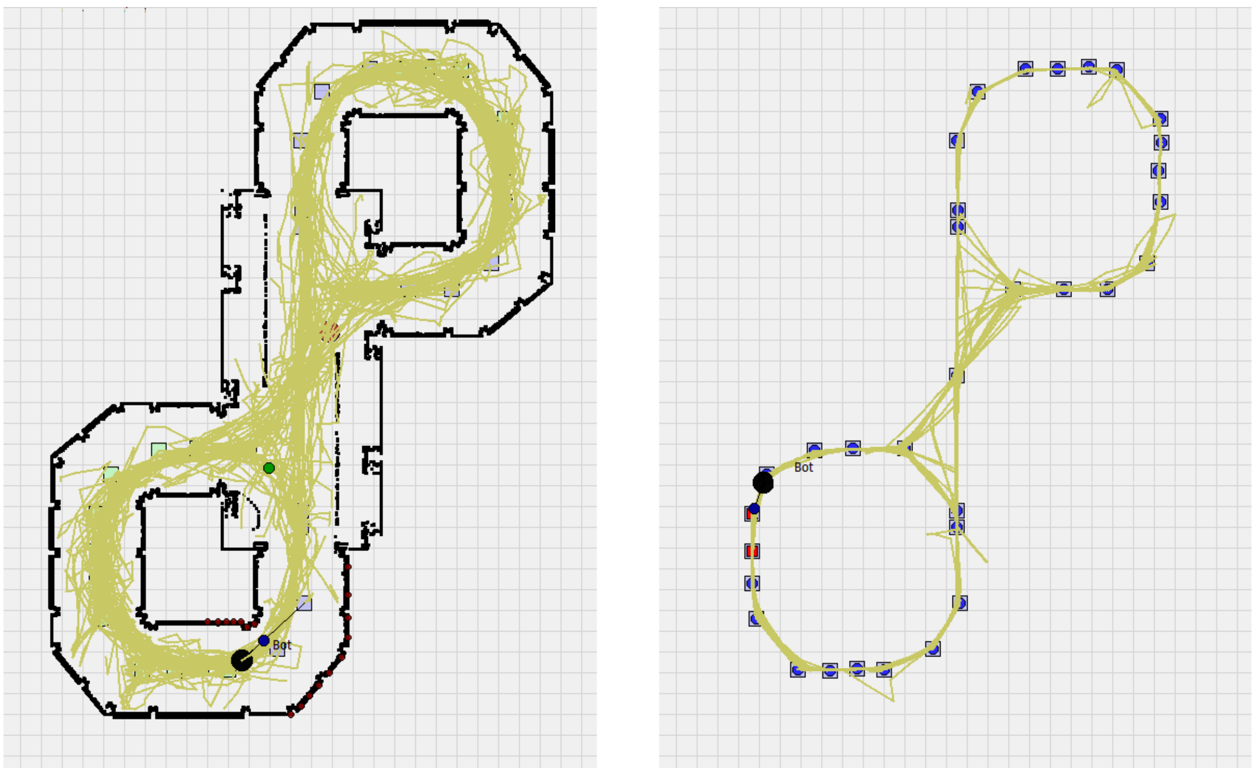


Figure 5.3: Left: Path followed by NeuroBot in a player VS player round. Right: Path followed by the common bot.

5.2 2K BotPrize Competition

5.2.1 Description

BotPrize [1, 20] is an annual bot-programming competition where research teams from different institutions around the world compete using their implemented bots. This competition evaluates the ability of these bots to provide human-like control of avatars within UT2004 environment. The sponsor of BotPrize which is the video games company 2K Australia, offers a major prize to the first team that will create a bot which is indistinguishable from human players and a minor prize to the winning team of the year. During the competition rounds, all bots and human judges play the game together and the latter rate the humanness of all the other avatars which might be controlled by other judges or bots. The bot that gathers the most points according to this judging system wins the competition. The point of the competition is that if a bot won the major prize then it might be considered to have human-level intelligence within the game environment according to Turing and his famous test [36].

The only regulation that a team should follow in order to be eligible is to create a bot for UT2004 that uses the GameBots2004 interface and runs on Windows. Although, the committee of the competition suggests using of Pogamut 3 integrated development environment (IDE), this is not a requirement. However, until 2011 no team had ever used a different IDE and the present work was the first that tried a different approach.

All things considered, the conditions of this competition can be considered ideal for testing of the performance of the present system and this is the reason that NeuroBot was one of the 2011 participations.

5.2.2 Results

Although NeuroBot was the only new participation for the year 2011, the outcome of the competition was fairly satisfactory with NeuroBot being characterized 35.7143 % human and thus achieving the second place. Indeed, NeuroBot had the highest score for the most part of the competition but was pipped by ICE team at the last minute. An important factor that influenced the results of the last rounds was that due to some internet connectivity issues, the speed at which the state of the neural network of NeuroBot was updating had to be decreased. Thus, during the last two out of a total of six rounds, NeuroBot became less responsive and thus lost the fist place.

The wining team managed to score slightly higher with an overall of 37.5% humanness whereas the other two teams that participated in the finals, Conscious-Robots and UT2, were ranked 26.6667% and 21.4286% respectively. However, once more, no team managed to create an agent whose behaviour can be considered indistinguishable from human players (i.e. score more than 50%). Detailed tables with the results of this competition can be found in the Appendix A.

Chapter 6

Conclusion

As a conclusion to this report, this chapter lists the key achievements of this project and provides a brief description of the planned next steps. Finally, a number of possible applications are also discussed.

6.1 Achievements

- The first major achievement of this work is the design and development of the first neural implementation of a global workspace architecture that is embodied in a dynamic real time environment.

- The second achievement is the implementation of a complete simulation framework which is general enough to simulate the behaviour of Multi-Agent Systems exploiting UT2004's features. This framework also has the ability to support architectures based on spiking neural networks (SNNs) and to visualize the performance of each architecture using various methods. Also, the integration of NeMo SNN simulation environment [14] makes this system capable of taking advantage of the available Graphics Processing Units (GPUs) and simulating large scale SNNs in real time.

- The third major achievement lies on the ability of this neural architecture to exhibit a level of human-like behaviour. This can be seen by taking into account the results of the 2K BotPrize 2011 competition where an agent using this implementation won the second place and its behaviour was characterized as 35.71% human, having a very small difference with the wining system that scored 37.5% (see Appendix A).

  Also, the time scale of this project was very short in contrast to the implementations of all the other teams, that have been participating in the BotPrize competition and advancing their systems for several years. Hence, the combination of these factors indicate that spiking neural networks can be considered as an advantageous way of producing human-like behaviour.

- The final major achievement is the acceptance of a paper [38] that described the neural architecture, for presentation and publication into the proceedings of the 2011 IEEE Conference on Computational Intelligence and Games in Seoul, South Korea. In this conference, the architecture was presented and discussed with AI experts from different institutions around the world and received positive comments.

6.2 Future Work

The next step for this work is the exploitation of a reinforcement learning technique, possibly using a Spike Time-Dependent Plasticity (STDP) rule. In a redesigned version of all the cortical modules, all the neurons of the module could be connected to each other before a training rule eliminates the unwanted connections and reinforces the right ones in order for the module to become functional.

For the better understanding of what is meant here, a good example is the "jumping" sensory module. At the present stage of the system, when "jumping" module gets activated, i.e. takes as input, the magnitude of the avatar's velocity vector that is parallel with the Z axis. If this activation is strong enough to win access to the workspace, this sensory module broadcasts the information that the avatar is jumping. However, different reasons may cause an increase in the vertical velocity of the avatar such as falling or using a lift. Hence, in a more advanced future version of the "jumping" module all three velocities could be taken as input and after a period of training the module could effectively distinguish whether the avatar is actually jumping or not.

This could be a very interesting extension, particularly for the behaviour and action selection modules, resulting a more abstract network architecture, with a good potential for an improved and more human-like performance.

A second interesting extension would be the exploitation of more biologically plausible input data such as the visual representation of the world in the same way that a real player can see it. This could be done by taking into account the raw image that comes from the game and processing it in real time in a way that is compatible with a neural network. The most biologically plausible approach would be to convert the image into a retina-like spiking representation. This type of processing of visual data has been used in iSpike C++ library [15]. Because of the computational complexity of this potential conversion, high performance parallel systems, such as a cluster of GPU processors along with compatible software [3], may be required. This addition would not change much of the current architecture because there is no need to keep the image representation in the working memory. The only part that would change is the sensory modules that would handle all the necessary processing in order to extract the aforementioned pieces of data from the incoming image in a way similar to the visual cortex.

Finally, another interesting extension to the present work would be the exploration via simulation of how neural synchronization could be used to implement a global workspace as proposed by Shanahan [33].

6.3 Possible Applications

An obvious practical application of this work would be the control of avatars in computer games. The majority of the available games today include an advanced dynamic environment where a human player plays against software agents. These games would be more challenging and rewarding if the opponents could display human-like behaviour. For this reason, several companies related to video games have shown great interest in such research with the 2K Australia being a characteristic example since it sponsors the BotPrize competition.

Another interesting application can be found in crowd simulators. These simulators are commonly used in public safety planning in order to develop crowd controlling strategies for emergency situations such as the avacuation of a building in the unfortunate event of fire or earthquake or even riots. They are also used in sociology research that is focused on social behaviour of people at social gatherings such as sort events, concerts or protests. In the above cases, understanding and simulating natural human behaviour under different types of stressful situations can lead to the creation of better models which can be used to develop better evacuation strategies or crowd controlling techniques.

The present system has been designed to be an autonomous controller of a human-like software agent that exists in a virtual environment governed by simulated laws of physics. It is based on the fact that low-level processes of a real human body are also handled by the simulation engine. Hence, noise factors, a very significant problem that can be found in all systems that interact with the real world, have been ignored. However, systems based on neural networks are well known for their robustness. Although it would be unlikely that the architecture of the neural network can control a real robot without major changes, the challenges in making this transition are less important that in cases of programmatic applications. Therefore, it is likely that a larger-scale version of this system with some necessary adjustments would perform well in a real environment controlling an embodied agent. In this case, variations of the system could be developed to control personal companion robots or future robotic assistants that are aimed at domestic applications.

All things considered, it is hoped that this work will be inspiration for the design of more advanced systems that would interact with our world in a similar way to ourselves.

Appendix A

Recorded Data

A.1 5-player rounds

Player	Exploration factor	Average health	Shooting time	Items taken
Human 1	0.62	69.2	136.00 sec	43.34
Human 2	0.58	74.69	271.84 sec	42.66
Human 3	0.49	64.66	155.12 sec	55.00
NeuroBot	0.43	74.16	200.66 sec	53.00
Common Bot	0.31	83.46	126.00 sec	173.20
	Average velocity	Number of jumps	Kills	
Human 1	17.69	86.80	6.3	
Human 2	13.00	48.00	9.6	
Human 3	12.00	49.66	34.6	
NeuroBot	7.02	96.00	20.1	
Common Bot	12.33	30.23	12.0	

Table A.1: The mean values of the measures that were used for the comparison of the behaviours in the first experiment

A.2 Player vs player rounds

Player	Exploration factor	Average health	Shooting time	Items taken
Human 1	0.617	93.33	68.93 sec	51.0
Human 2	0.699	105.62	217.52 sec	88.8
Human 3	0.633	89.01	75.84 sec	26.2
NeuroBot	0.483	76.32	152.43 sec	85.1
Common Bot	0.334	88.10	49.00 sec	173.6
	Average velocity	Number of jumps	Kills	
Human 1	15.71	74.9	18.9	
Human 2	7.80	45.4	33.8	
Human 3	12.81	51.0	79.0	
NeuroBot	10.49	168.4	25.8	
Common Bot	8.66	17.1	19.1	

Table A.2: The mean values of the measures that were used for the comparison of the behaviours in the second experiment

A.3 BotPrize 2011

Most human bots		Most human humans		Most human epic bots	
Bot name	Humanness	Player name	Humanness	Skill level	Humanness
UT^2 doozy	44.4444%	Sisko	100.0000%	3	57.1429%
NeuroBot	40.0000%	array2001	80.0000%	4	25.0000%
UT^2 server	23.0769%	phi	57.1429%	2	0.0000%
Conscious-Robots	0.0000%	ikon2	33.3333%		
ICE-CIG2011 Murakami	0.0000%				

Table A.3: First trials for 2K Botprize 2011 humanness results

Best bot judges		Best human judges	
Bot name	accuracy	Human name	accuracy
ICE-CIG2011 Murakami	100.0000%	ikon2	81.8182%
UT^2 server	59.2593%	Sisko	77.7778%
NeuroBot	53.8462%	ashida	70.0000%
UT^2 doozy	29.1667%	phi	65.5172%
		array2001	57.1429%

Table A.4: First trials for 2K Botprize 2011 judging results

Most human bots		Most human humans		Most human epic bots	
Bot name	Humanness	Player name	Humanness	Skill level	Humanness
NeuroBot	44.4444%	dan	71.4286%	1	50.0000%
UT^2 doozy	42.8571%	phi	70.0000%	2	18.1818%
Conscious-Robots	33.3333%	Sisko	66.6667%	3	0.0000%
ICE-CIG2011	33.3333%	I went rrr	14.2857%	4	0.0000%
UT^2 server	0.0000				

Table A.5: Second trials for 2K Botprize 2011 humanness results

Best bot judges		Best human judges	
Bot name	accuracy	Human name	accuracy
UT^2 server	64.7059%	Sisko	64.7059%
NeuroBot	57.1429%	I went rrr	64.5161%
UT^2 doozy	36.3636%	dan	62.8571%
ICE-CIG2011	34.7826%	phi	33.3333%

Table A.6: Second trials for 2K Botprize 2011 judging results

Most human bots		Most human humans		Most human epic bots	
Bot name	Humanness	Player name	Humanness	Skill level	Humanness
ICE-CIG2011	37.5000%	KyeongJong	66.6667%	1	50.0000%
NeuroBot	35.7143%	dan	60.0000%	2	50.0000%
Conscious-Robots	26.6667%	HyunsooPark	50.0000%	5	41.6667%
UT^2	21.4286%	someplayer	50.0000%	4	33.3333%
		Sisko	20.0000%	3	33.3333%
				All	40.0000%

Table A.7: 2K Botprize 2011 humanness results

Best bot judges		Best human judges	
Bot name	accuracy	Human name	accuracy
Conscious-Robots	50.0000%	someplayer	66.6667%
ICE-CIG2011	50.0000%	HyunsooPark	66.6667%
NeuroBot	42.8571%	KyeongJong	58.8235%
UT^2	41.6667%	dan	52.6316%
		Sisko	47.6190%

Table A.8: 2K Botprize 2011 judging results

Appendix B

User Guide

Installation

NeuroBot simulator comes with an automatic Windows installer which unpacks all the necessary files and libraries to the desired folder. However, if the source code is provided, NeuroBot simulator can be built on every platform that supports the Qt framework, the NeMo simulator library and the Eigen library.

The operating systems that have been tested are:

- Windows XP (or newer)
- Ubuntu 8.10 (or newer)

System Requirements

NeuroBot simulator can automatically detects if the system in which it has been installed has compatible hardware in order to run the simulations using multiple cores of the graphics processing unit (GPU) concurrently. If this is not the case, then it runs the simulation using CPU. All NVIDIA graphics cards that support CUDA drivers, newer than version 2.9, are considered compatible. However, NeuroBot simulator can support this feature, only if NeMo simulator has been installed to this system too.

Supported Modes

In the initial screen of the program, the user is called to decide which mode they want to use. The available modes are tree:

Bot connection mode. In this mode, the program becomes a client that supports bot connections with GameBots2004 using TCP/IP protocol and GameBots communication protocol. To use this mode, the user needs to specify the IP in which the GameBots server is running, the port that this server uses for a bot connection, the desired user name and in some cases a password. If the connection is successful the user is able to see the console window that shows the incoming and out coming messages as well as the map window that illustrates the system's internal representation of the Unreal Tournament 2004 (UT2004) environment.

Observer connection mode. In this mode, the program again becomes a client of a GameBots2004 server in order to support observer connections. To use this mode, the user needs to specify the IP of the server, the port that the server uses for observer connections, a potential password and the name of the player that they want to observe. If the connection is successful and there is a player with this name, the user is again able to see the console window, as well as a more general version of the map window that illustrates all the players that are currently on the game and provides real-time statistics for the selected player.

Debugging Mode. This mode does not require an active GameBots2004 connection since the simulation here is independent of the UT2004 environment.

Configuration File

The file "config.ini", that can be found in the main path of the program, is used to allow users to alter some parameters of the execution of the program as well as to change default values. Table B.1 provides a brief description and the default value of each parameter included on this file.

Parameter	Default value	Description
TimeInterval	5	How many milliseconds between two updates of the state of the network.
DefaultIP	195.242.237.18	The IP of the GameBots server.
BotConnectionPort	3000	The port that GameBots server uses for the desired connection
PlayerName	NeuroBot	The desired name of the bot or the observed player
NemoDiagrams	false	If the available graphs will be visible.
AutoLoadMap	false	If NeuroBot simulator will automatically retrieve a saved map.
MapFile	map.sav	The name of the file that in which the first map will be saved
PathFile	path.sav	The name of the file that in which the first path will be saved
MapGridSize	45	How much area of the screen is covered by the map window
MapGridWidth	20	Zoom of the map
UseFileNumbers	true	If this is true, each map, path and statistics file will be saved on a different file

Table B.1: The parameters of the configuration file.

Hot-Keys

Key	Description
ESC	The program exits.
L	Save map.
K	Load map.
M	Save path.
N	Load path.
C	Place the bot in the centre of the window.
V	Auto-centring bot.
5	Save statistics.
A,S,D,W	Change position of the map.

Table B.2: Map window hot-keys.

Key	**Description**
ESC	The program exits.
W	Move the avatar forward.
S	Move backwards.
A	Move left.
F	Move right.
Q	Turn left.
E	Turn right.
Space	jump.
Alt	Start/stop firing.
R	Choose best weapon.

Table B.3: Console window hot-keys.

Key	**Description**
ESC	The program exits.
Space	Start/stop real time simulation.
J	Start/stop producing and illustrating graphs.
5	Braitenberg Vehicle simulation mode.
8	Neuronal global workspace simulation mode.
O	Add stimulus of a scalar value.
I	Add stimulus of a vector value.

Table B.4: NeMo window hot-keys.

Bibliography

[1] 2k botprize:. http://botprize.org/index.html.

[2] 2k botprize won by a 'conscious' bot:. http://aigamedev.com/open/articles/conscious-bot/.

[3] Nemo neural simulator:. http://www.doc.ic.ac.uk/ akf/nemo/index.html.

[4] Unrealscript programming language:. http://unreal.epicgames.com/UnrealScript.htm.

[5] *Machine Learning*. McGraw Hill, 1997.

[6] John Anderson. *Rules of the mind*, pages 3–4. Lawrence Erlbaum Associates, 1993.

[7] B.J. Baars. The conscious access hypothesis: Origins and recent evidence. pages 47–52.

[8] B.J. Baars. A cognitive theory of consciousness. pages 27–51, 1988.

[9] B.J. Baars. In the theater of consciousness: The workspace of the mind. 1997.

[10] A. Bouganis and M. Shanahan. Training a spiking neural network to control a 4-dof robotic arm based on spike timing-dependent plasticity. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1 –8, july 2010.

[11] Valentino Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. Cambridge, MIT Press.

[12] Stanislas Dehaene and Jean-Pierre Changeux. Ongoing spontaneous activity controls access to consciousness: A neuronal model for inattentional blindness. *PLoS Biol*, 3(5):e141, 04 2005.

[13] Stanislas Dehaene, Claire Sergent, and Jean-Pierre Changeux. A neuronal network model linking subjective reports and objective physiological data during conscious perception. *Proceedings of the National Academy of Sciences*, 100(14):8520–8525, 2003.

[14] A.K. Fidjeland and M.P. Shanahan. Accelerated simulation of spiking neural networks using gpus. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1 –8, july 2010.

[15] Gamez D. Lazdins E. Fidjeland, A. and M. Shanahan. ispike: A spiking neural interface for the icub robot. In *Poster presented at the International Workshop on Bio-Inspired Robots, Nantes, France.*, 2011.

[16] Stan Franklin. Ida a conscious artifact? *Journal of Consciousness Studies*, 10(45):4766, 2003.

[17] David Gamez. Information integration based predictions about the conscious states of a spiking neural network. *Consciousness and Cognition*, 19(1):294 – 310, 2010.

[18] Jakub Gemrot, Rudolf Kadlec, Michal Bda, Ondej Burkert, Radek Pbil, Jan Havlek, Luk Zemk, Juraj imlovi, Radim Vansa, Michal tolba, Tom Plch, and Cyril Brom. Pogamut 3 can assist developers in building ai (not only) for their videogame agents. In Frank Dignum, Jeff Bradshaw, Barry Silverman, and Willem van Doesburg, editors, *Agents for Games and Simulations*, Lecture Notes in Computer Science, page 1. Springer Berlin / Heidelberg.

[19] H. Hagras, A. Pounds-Cornish, M. Colley, V. Callaghan, and G. Clarke. Evolving spiking neural network controllers for autonomous robots. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 5, pages 4620 – 4626 Vol.5, april-1 may 2004.

[20] P. Hingston. A turing test for computer game bots. *Computational Intelligence and AI in Games, IEEE Transactions on*, 1(3):169 –186, sept. 2009.

[21] A. Hodgkin and A. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. 1952.

[22] Eugene Izhikevich. Simple model of spiking neurons. 2003.

[23] Eugene Izhikevich. Which model to use for cortical spiking neurons. 2004.

[24] Jeffrey L. Krichmar, Douglas A. Nitz, Joseph A. Gally, and Gerald M. Edelman. Characterizing functional hippocampal pathways in a brain-based device as it solves a spatial memory task. *Proceedings of the National Academy of Sciences of the United States of America*, 102(6):2111–2116, 2005.

[25] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659 – 1671, 1997.

[26] Colin M. MacLeod. Half a century of research on the stroop effect: An integrative review. *Psychological Bulletin*, 109(2):163 – 203, 1991.

[27] Araceli Sanchis Raul Arrabales, Agapito Ledezma. Cera-cranium: A test bed for machine consciousness research. 2009.

[28] Jane E. Raymond, Kimron L. Shapiro, and Karen M. Arnell. Temporary suppression of visual processing in an rsvp task: An attentional blink? *Journal of Experimental Psychology: Human Perception and Performance*, 18(3):849 – 860, 1992.

[29] Andrew Scholer Sheila Tejada Rogelio Adobbati, Andrew N. Marshall. Gamebots: A 3d virtual world test-bed for multi-agent research. 2001.

[30] Frank Rosenblatt. The perceptron–a perceiving and recognizing automaton. Technical report, Cornell Aeronautical Laboratory, 1957.

[31] Murray Shanahan. A cognitive architecture that combines internal simulation with a global workspace. *Consciousness and Cognition*, 15:433–449, 2006.

[32] Murray Shanahan. A spiking neuron model of cortical broadcast and competition. *Consciousness and Cognition*, 17(1), 2008.

[33] Murray Shanahan. *Embodiment and the inner life: cognition and consciousness in the space of possible minds.* Oxford University Press, 2010.

[34] Jean-Pierre Changeux Stanislas Dehaene, Michel Kerszberg. A neuronal model of a global workspace in effortful cognitive tasks. page 14529, 1998.

[35] Giulio Tononi. An information integration theory of consciousness. *BMC Neuroscience*, 5(1):42, 2004.

[36] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 10 1950.

[37] Stan Franklin Uma Ramamurthy. Self system in a model of cognition. page 514, 2011.

[38] Andreas Fidjeland Zafeirios Fountas, David Gamez. A neuronal global workspace for human-like control of a computer game character. *IEEE Conference on Computational Intelligence and Games (to appear)*, 2011.

[39] Ariel Zylberberg, Diego Fernndez Slezak, Pieter R. Roelfsema, Stanislas Dehaene, and Mariano Sigman. The brain's router: A cortical network model of serial processing in the primate brain. *PLoS Comput Biol*, 6(4):e1000765, 04 2010.