CONTENTS

1 INTRODUCTION

1.1 COMPUTER GAMES AND ARTIFICIAL INTELLIGENCE

The term *computer games* refers to interactive games operated by computer circuitry. The platforms on which computer games are played include personal computers, arcade consoles, video consoles and handheld devices [1].

It is believed that the first electronic game was created by William A. Higinbotham – of the Brookhaven National Laboratory, in 1958, as interactive technology demonstration available for laboratory's visitors. *Tennis for Two* was working on analog computer and was simulating a tennis game on an oscilloscope. It was presenting a simplified projection of a tennis court from the side, featuring a gravity-controlled ball that needed to be played over the net. The players used analog controllers to adjust the trajectory of the ball and a button to hit it with an invisible racket [2].

Since then, computer games evolved from small programs or devices developed by individuals to major commercial projects produced by teams of experienced developers working, in some cases the process lasted for years on a particular product. The whole industry is estimated to be worth $11.7 billion in 2008 just in United States, which places it in front of music and film industries [3].

To be continued …

1.2 FIRST PERSON SHOOTER GAMES

1.3 MACHINE LEARNING AND COMPUTER GAMES

1.4 THESIS OVERVIEW

2 BACKGROUND

In the first chapter we showed … importance of the topic.

In the beginning of this chapter we introduce AI in First-Person Shooter games, presenting also some AI methods which are commonly used to accomplish bot's tasks. Following, we present the summary of the Machine Learning techniques used in computer games. Next section depicts the concept of Reinforcement Learning and introduces the Connectionist Q-Learning algorithm. Finally, the last section describes ID Software's Quake II as an example of an FPS game and QASE API – both used in the practical part of this thesis.

2.1 ARTIFICIAL INTELLIGENCE IN FIRST-PERSON SHOOTER GAMES

2.1.1 INTRODUCTION

First-Person Shooter games (commonly abbreviated to FPS games), are one of the most popular genres of computer games. In FPS, human players use a mouse and a keyboard to control their virtual in-game character. The main input for a player is the first-person perspective view of the world displayed on the screen and sounds played in the game. The usual scenario in an FPS game focuses on fighting against opponents using some sort of firearms. The player's character is placed in the three dimensional world together with other opponents, which can be controlled by other human players or by computer programs called *bots*[1].

All participants of the game can move around the world and pick up weapons and special items such as medical kits and armor jackets. Each FPS game is different, but usually, player's health is described with some number and, if player's health is poor, it can be recovered with a medical kit. If a player wears an armor jacket, the damage taken from gunshots will be reduced.

The form of the game determines a set of basic actions that all the players need to perform. This includes navigating through the three dimensional world, selecting an appropriate item or gun to use, aiming and shooting at the enemies. In a perfect case in a result of performing optimal actions, the player should win the game[2]. In practice, games also have some random factors, that make them less predictable and more difficult to control.

The best human players are still better than the best bots. Although computers have potential to be better at some actions, like aiming and shooting accurately, in most of complex FPS

[1] Bot - a habitual, shorter form of robot. The term bot is also used when referring to computer programs working on the Internet performing repetitive actions, such as price comparison, searching information etc.
[2] Winning conditions of a game depend on a particular game scenario – it may mean, for instance, defeating all the enemies without hurting hostages. Depends on a particular game.

games less precise human players still manage to develop tactics which allow them to win. Philip Hingston [4] proposes a variant of Turing Test designed for FPS bots on which the *BotPrize* competition is based. In the competition taking place every year, the human players play with bots an FPS game, while being observed by judges. Basing only on observed behavior of game characters, the judges have to tell the human players from bots. Till now, none of bots have managed to appear human-like enough to win the *BotPrize*, but also all of them have been losing the game with human players [5].

2.1.2 BOTS ARCHITECTURE

The list of basic actions, that a player needs to perform in each game can be used to develop a generic architecture of FPS AI. Paul Tozour [6] proposes an architecture divided into four main components: animation, movement, combat and behavior. Figure 1 presents a diagram representing those four basic components.

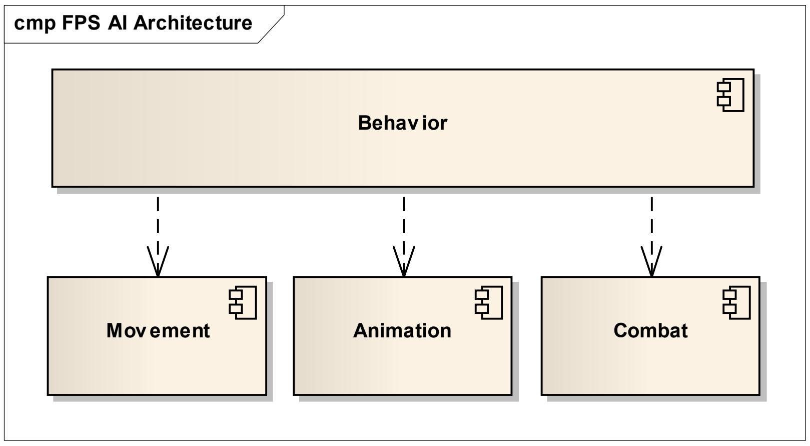

Figure 1: UML component diagram of an FPS game AI architecture proposed by Paul Tozour. A dashed arrow represents a functional dependency between components.

The animation component is responsible for controlling the character's virtual body. This can be done by adjusting parameters of existing animations (e.g. character's running speed), playing a right animation at a right time (e.g. climbing up the ladder) or by solving an inverse kinematics problem[3], when a character reaches for an item. This component should also control which parts of the body are performing which animation and deal with conflicts (e.g. bot death animation should have higher priority than bot jump animation).

Bot's movement or navigation controller provides a service for other components – it allows them to move a bot from its current position to a specified one. This task requires a bot to perform path finding. It has to decide following which path it will move towards its destination. The path is usually represented as a sequence of points in the world, that a bot has to follow, which involves using some abstract representation of the game world, called a map. In the next chapter we will take a closer look at common game map representations. After the path has been established, the movement controller turns the character in the right direction

[3] The inverse kinematics problem can be stated as a question: Given the desired position of the robot's hand, what should be the angles at all robot's joints? The forward kinematics problem seeks at what position will be robot's hand providing the given angles at robot's joints.

and controls its movement from one point of the path to another. Also, if some dynamic obstacles appear, the movement controller should respond appropriately – trying to solve the problem or reporting it.

When a bot enters combat, the combat controller should take over the control of most of bot's behaviors, such as weapon and opponent selection, firing and maneuvering or picking up items. The main challenge here is to quickly evaluate a situation and choose an appropriate tactic, which shows up to be quite easy for humans and difficult for computers. One reason for that may be that we are very good at evaluating the spatial configuration of entities in the world, which allows us to take better decisions. For instance, humans quickly find good places to hide from a gunfire or to shoot at the enemy. Modern bots still find this task difficult and base on scripted, pre-defined by their authors behavior. Another aspect of combat, that the combat component should control is the group tactics and communication between group members while in combat.

The behavior component is one that controls all the other components and takes high-level decisions about bot's behavior. It decides whether the bot should search for an enemy or a better weapon, whether it should enter into combat or retreat. As this is a managing controller, it's quality will determine bot's resulting behavior.

2.1.3 NAVIGATION SOLUTIONS

The spatial reasoning cannot be performed on the raw geometry of the game world. The main reason is complexity. A single brick in a wall can be described with as many as thousands of polygons, with a wall consisting of hundreds of bricks. What a bot needs to know is that there is a wall. All the additional information is not important when performing a path finding. It makes the task computationally expensive, while a bot needs to operate in a real-time. More abstract representation of the game world is necessary.

Regardless of chosen representation, most of modern computer games have it prepared by their creators (like in [7] and [8]) before the game is released. Although works like [9] try to make this process automatic, bots are not yet able to learn about the world by themselves, at least not well and fast enough to satisfy game developers' requirements.

Waypoint map

One of the most popular abstract game world representation is a waypoint map. Generally speaking, a waypoint map is a graph in which nodes represent reachable points in the game world, and the edges indicate that it is possible to move from one node to another.

The edges can be marked with a distance or with an action necessary to take in order to move from one node to another (e.g. jump or crouch). The nodes, on the other hand, can also contain some additional information, like an item type that can be expected in a given place or an information that the given node is a good place to hide at.

It is important to make sure that moving from one node to another that is connected with an edge can be easily performed by a navigation module. Usually it means, that all the bot needs to do is to turn towards a destination waypoint and move forward until it arrives there.

Having such a representation of the game world, we can easily navigate between any points on the waypoint map if an appropriate path exists. To perform path finding one of the graph search algorithms can be used, or if the game environment is static enough, all the paths can be computed before the game. However, it is important to make sure that the path finding works fast enough for a real-time game.

Navigation mesh

In recent years, the navigation mesh has become the world space representation of choice for agents in virtual worlds [9]. It divides all the walk-able surfaces of the environment into convex polygons, creating what can be called a “floor plan” of the world. Navigation mesh can also be represented as a graph in which nodes are polygons, and the graph edge exists between two nodes if their polygons have a common edge.

Navigation meshes are considered to be more powerful and providing more realistic navigation [10]. In a waypoint map, a bot could be located only at the waypoints or somewhere on the edge between them. In navigation mesh a bot can walk over the whole surface of each polygon. This allows more flexible, less schematic and more realistic movement, while still being relatively simple representation of the game map. Since we still use a graph, the path finding can be performed in exactly the same way as in case of waypoint maps.

The difficult part in the navigation mesh is how a bot should move from one polygon to another. This may require not only finding an appropriate polygon’s edge and moving towards it, but also avoiding dynamic obstacles that may appear on the way. In case of a non-player characters that do not live long enough in the game, being usually shot by a player, it may not be cost-efficient to develop a navigation mesh based movement component. But if a human player will have enough time to take a closer look at our bot, the navigation mesh is a better choice.

2.1.4 FINITE STATE MACHINES

State machines, along with scripting are two most common techniques used in modern games to perform decision-making. Their popularity can be a result of simplicity and the power of expression.

Often game characters will behave in a certain way until some event occurs. For instance, a bot will search for a weapon, but as soon as it sees an enemy it should change its behavior and decide whether to fight or retreat. This kind of behavior can be supported with finite state machines (FSM).

An FSM is a system that has a limited number of states. At a moment only one state is occupied. Each state can be associated with some specific bot’s action. Transitions exist between states. Each transition has a set of associated conditions. If conditions of a transition are met, the machine moves from one state to another. An example of game FSM is presented on Figure 2. In this example, for instance, if a bot is in state "Search for enemies and fight" and it gets wounded during a fight, the transition to the state "Search for a medical kit" takes place and the bot starts to search for a medical kit in order to heal itself.

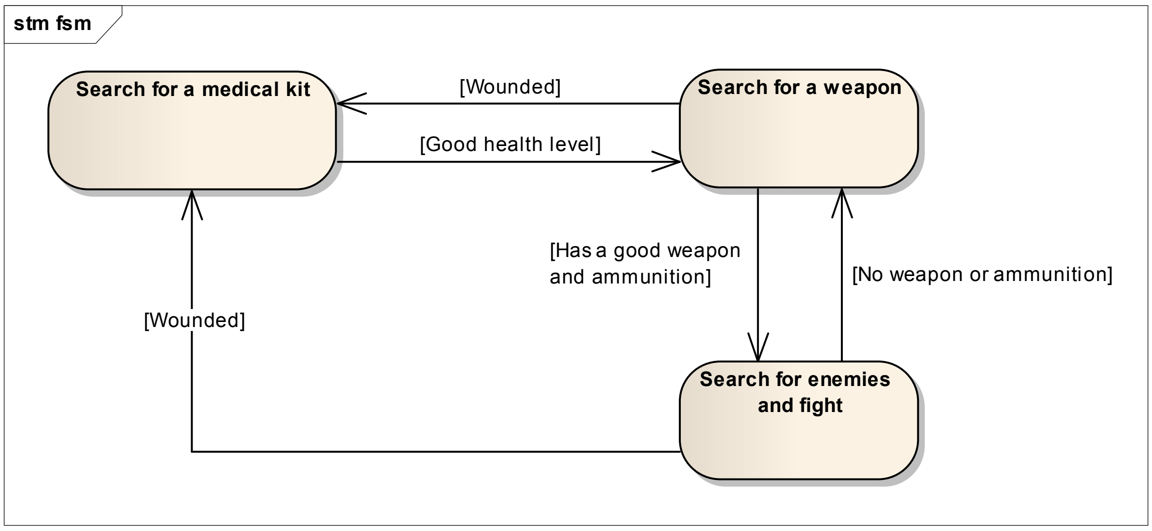
stm fsm
Search for a medical kit
[Wounded]
[Good health level]
Search for a weapon
[Has a good weapon and ammunition]
[No weapon or ammunition]
[Wounded]
Search for enemies and fight

Figure 2: An example of a simple game FSM in UML notation.

Finite state machines are easy to use and read. A designer can also, without too much trouble, manage a level of detail in which the model reflects a desired behavior by adding or removing additional states and transitions.

2.1.5 FUZZY LOGIC

In a narrow sense, fuzzy logic is a logical system, being a generalization of conventional logic. While in conventional logic a variable can either be true or false, in fuzzy logic it can have associated any real number in a range $[0$. The 0 value is interpreted as false and 1 as true. The numbers in between express the "degree of truth".

Fuzzy logic in a wide sense is a term that refers to a union of fuzzy logical system, fuzzy set theory, possibility theory, calculus of fuzzy if-then rules, fuzzy arithmetic, fuzzy quantifiers and all other theories derived from the concept of fuzzy logical system [11]. Most of these theories provide tools that can be used when taking decisions with estimated values under incomplete or uncertain information.

There have been many successful applications of fuzzy logic in signal processing, pattern recognition, business forecasting, speech processing, robotics control, natural language understanding etc. Computer games are not an exception – fuzzy logic has been used in many commercial games. It is attractive because of speed of calculation and ability to model complex behavior [11] [12].

It is also often used in combination with other techniques, like presented in [13] FuSM – Fuzzy State Machine. For an FPS bot, fuzzy sets and relations can be useful to express how much a bot wants to do or have something. In the following paragraphs, some basic theory behind fuzzy relations is recalled.

Fuzzy sets

Let X be a nonempty set. A fuzzy set A in X is characterized by a function:

$$\mu_A(x) : X \to [0;1] \tag{1}$$

$\mu_A(x)$ is called a membership function and expresses the degree of membership of an element x in a fuzzy set A. 1 is equivalent of classical truth, whilst 0 is false. We can note that a fuzzy set A is fully determined by the set of pairs:

$$A = \{ (x, \mu_A(x)) : x \in X \} \tag{2}$$

The fuzzy sets theory is a generalization of classical sets theory. If we were to define a classical set B using a fuzzy set theory, our membership function could be defined as follows:

$$\mu_B(x) = \begin{cases} 1 \; if \; x \in B \\ 0 \; otherwise \end{cases} \tag{3}$$

Fuzzy relations

Let's consider a classical relation. Let $X_1, X_2, \dots, X_n$ be classical nonempty sets. An n-ary relation R is a subset of Cartesian product of n nonempty sets:

$$R \subseteq X_1 \times X_2 \times \dots \times X_n \tag{4}$$

Since a relation is a set, we can use a membership function to define a fuzzy relation. For a classical n-ary relation a membership function is defined as follows:

$$\mu_R(x_1, x_2, \dots, x_n) = \begin{cases} 1 \; if \; (x_1, x_2, \dots, x_n) \in R \\ 0 \; otherwise \end{cases} \tag{5}$$

Whilst for a fuzzy n-ary relation a membership function can take all values from a range $[0;1]$.

For example a binary relation called "approximately equal", defined on set N^2, where N is a set of natural numbers, could be defined with a following membership function:

$$\mu(n,m) = \begin{cases} 1 \; if \; n = m \\ 0.7 \; if \; |n-m| = 1 \\ 0.3 \; if \; |n-m| = 2 \\ 0 \; otherwise \end{cases} \tag{6}$$

To use fuzzy relations in bot programming, we usually need also some operations on them. The most basic operations on fuzzy relations are intersection and union.

Intersection and union operations

Let R and S be binary relations defined on set $X \times Y$. The intersection of R and S is defined as follows:

$$(R \wedge S)(u,v) = \min\{R(u,v), S(u,v)\} \tag{7}$$

The union of R and S is defined as follows:

$$(R \vee S)(u,v) = \max\{R(u,v), S(u,v)\} \tag{8}$$

To illustrate operations on fuzzy relations let's suppose we have bots set B and items set I. Let fuzzy relation N be defined as "bot needs item" and fuzzy relation C be defined as "bot is close to item". Both relations are defined on Cartesian product $B \times I$.

If we want to choose an item that is close to a particular bot *and* the bot needs it, we will choose an item that has a highest fuzzy relation of an intersection $R \wedge$. If, on the other hand we are interested in an item that is close *or* a needed item, we will use a union operation.

Exhaustive compilation of many other possible fuzzy operations as well as more advanced fuzzy reasoning study can be found in [11].

2.1.6 SCRIPTING

2.2 MACHINE LEARNING IN COMPUTER GAMES (LEAVE IT? REMOVE IT?)

2.2.1 INTRODUCTION

2.2.2 ONLINE AND OFFLINE LEARNING

2.2.3 TESTING ISSUES

2.3 REINFORCEMENT LEARNING

2.3.1 Introduction

2.3.2 ??? – more information

2.3.3 Connectionist Q-learning

2.3.4 Applications

2.3.5 Related work

2.4 Quake II and QASE API

2.4.1 Introduction

2.4.2 Game choice

2.4.3 QASE API

2.4.4 Related work

3 PROJECT REQUIREMENTS

3.1 PROJECT SCOPE

There are two main objectives of the practical part of this thesis:

1. Design and develop Quake II bot that is able to compete with human players and other third-party bots.
2. Apply on-line reinforcement learning methods to chosen bot's decision problem.

The first one will allow us to test in practice basic methods of FPS AI programming and will provide a base for further development.

Completing the second objective will let us evaluate suitability of on-line reinforcement learning in this application as well as its implementing and testing complexity, which is especially interesting for commercial game developers.

Whether a bot is able to compete with human players and third-party bots will be verified with an experiment under conditions similar to a typical Quake II match.

3.2 ASSUMPTIONS

In this section basic assumptions of the practical part of the thesis are described.

A typical commercial bot AI is a complex program, for instance [7] or [8], and it is usually developed by a small group of people. This to a degree motivates the following assumptions, as having limited resources, the author had to focus on chosen, more specific aspects of bot development.

QASE API

QASE API (described in chapter XXX) allows rapid development of Quake II bots focusing on AI rather than on game-specific issues like communication with a game server.

Navigation module

The navigation module will be kept as simple as possible while still being effective enough to compete with other bots and human players.

In order to achieve it, the QASE API's waypoint map generation module will be used to facilitate the process of supplying the bot with a knowledge about a game map.

Main focus

It is assumed that this thesis will focus on the combat skills of programmed bots.

In the early stage of the work, the author has noticed that it is relatively difficult to evaluate the quality of a particular bot's navigation or general decision-making skills. At the same time the evaluation of bot's combat skills can be quite direct, for instance, by measuring the inflicted damage to the opponent.

3.3 LIMITATIONS

The main limitations of the project are determined by QASE API - all developed bots will be client-side bots.

Quake II server in order to keep all the clients up to date with the world information sends new messages to each client every 100 milliseconds. This creates a delay between bot's perception and actions execution. For instance, if bot receives the information about enemy's position at time *t*, and decides to shoot at him, the shooting will take effect at time *t+1*, at which the enemy may already be at different position.

On the other hand, a server-side bot may be programmed in such a way that it will react and perceive at the same, from other client's point of view, time. This gives an important advantage to a server-side bot.

Another aspect of client-side bots is that the server informs them only about the world in their immediate surroundings, while the server-side bot can have access to full knowledge about the current game state. This includes positions of all the enemies and availability of items even in the most remote parts of the map, which again, makes it easier for a server-side bot.

Although the development of a client-side bot has more constraints, it is worth noticing that from a human player's point of view the client-side bot may be seen as a more fair opponent in comparison with a server-side one.

4 DEVELOPED SOLUTIONS

4.1 INTRODUCTION

4.2 REFERENCEBOT

4.3 RLBOT

4.4 ERASERBOT

4.5 SUPPORT APPLICATIONS

5 EXPERIMENTS DESCRIPTION

6 Results

7 Conclusions

8 ATTACHMENTS

9 BIBLIOGRAPHY

[1] Encyclopaedia Britannica, inc., *The New Encyclopaedia Britannica.*, Chicago: Encyclopaedia Britannica, 2007.

[2] J. Gettler, "The First Video Game?."

[3] Entertainment Software Association, *2009 Sales, demographic and usage data*.

[4] P. Hingston, "A Turing Test for Computer Game Bots," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, 2009, s. 169-186.

[5] "Botprize contest homepage."

[6] P. Tozour, "First-Person Shooter AI Architecture," *AI Game Programming Wisdom*, 2002.

[7] J. van Waveren, "The Quake III Arena Bot," Delft University of Tehchnology, 2001.

[8] R. Straatman, W. van der Sterren, i A. Beij, *Killzone's AI: dynamic procedural combat tactics*, Amsterdam: Guerilla Games, 2005.

[9] D.H. Hale, G.M. Youngblood, i N.S. Ketkar, "Using Intelligent Agents to Build Navigation Meshes," 2010.

[10] P. Tozour, "Building a Near-Optimal Navigation Mesh," *AI Game Programming Wisdom*, 2002.

[11] C. Alavala i ebrary, Inc., *Fuzzy logic and neural networks basic concepts and application*, New Delhi :: New Age International (P) Ltd., Publishers,, .

[12] M. Zarozinski, "An Open-Source Fuzzy Logic Library," *AI Game Programming Wisdom*, 2002.

[13] M. DeLoura, *Game programming gems 2*, Hingham Mass.: Charles River Media, 2001.