# 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.

## Artificial Intelligence in First-Person Shooter games

### Introduction

First-Person Shooter games (commonly abbreviated to FPS games), are one of the most popular genre of computer games. In FPS, human players use mouse and a keyboard to control their virtual in-game character. They base on 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]](#footnote-2).

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]](#footnote-3). 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 games less precise human players still manage to develop tactics which allow them to win. Philip Hingston [1] proposes a Turing Test variant 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 the 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, but also all of them have been losing the game with human players [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 [3] 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]](#footnote-4), 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 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.

### 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.

#### 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, 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 [4]. 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 [5]. 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.

### Finite State Machines

### Fuzzy Logic

### Scripting

## Machine Learning in computer games (Leave it? Remove it?)

### Introduction

### Online and offline learning

### Testing issues

## Reinforcement Learning

### Introduction

### ??? – more information

### Connectionist Q-learning

### Applications

### Related work

## Quake II and QASE API

### Introduction

### Game choice

### QASE API

### Related work

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. [↑](#footnote-ref-2)
2. Winning conditions of a game depend on a particular game scenario – it may mean, for instance, defeating all the enemies without hurting hostages. It depends on a particular game. [↑](#footnote-ref-3)
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. [↑](#footnote-ref-4)