MAS Project

TI1606 Final Report

Team

Authors

Youp Mickers # 4246713
Joshua Slik # 4393007
Ashay Somai # 4366220

Gavin Sun # 4412249
Jelle Tjallema # 4215257
Lisette Veldkamp # 4394712

Contents

Part I

Report

Preface

Introduction

Problem Unreal Tournament 3 is a game with a simple objective: Capture the enemy flag. However simple this objective may seem and however trivial a task it can be for a human, this task is already quite extensive for a machine.

Background Unreal Tournament 3 is the third instalment in the Unreal Tournament series, a series of so called *First Person Shooter* games. It has multiple modes of games: Death Match, killing as many enemies as you can. Team Death Match is the same, only with teams instead of everybody versus everybody. The last is Capture the Flag, in which there are two teams, each with a flag. The objective is to walk to your enemies' flag, pick it up and bring it back to your flag to capture it. We will focus on the latter category.

Main Question Our objective is to write a so-called *agent* which is capable to defeat computer controlled enemies written by the game's programmers. Our team will consist of four agents who will talk to each other to maximise their efficiency. Our team will take on a team of three enemies of the highest possible level.

Other Questions

Structure

Summary

Chapter 1

Introduction

Chapter 2

Program of requirements

2.1 Prioritising Requirements

Using the MoSCoW method, where the M stands for Must, the S for Should, the C for Could and the W for Won't, we decided to put all the minimum requirements under the M. All the other functionalities, which are use to better our bots, are placed under the S. Everything that was taken into consideration, but eventually denied, because it was to hard to implement, was placed under the C. The rest are placed under the W.

2.2 Functionalities

2.2.1 Minimum Functionality

The minimum functionality is basically to grab the enemy flag and return it. And eventually you should win the game of Capture the Flag. This taken to account, we decided to make it our number one priority. We also knew that by only implementing this, we would definitely lose. Maybe in the beginning, when there is only one level 1 epic bot, we would have a chance at beating them. To make successful bots we should have additional functionalities.

2.2.2 Advanced Functionality

To extend our bots, we have divided the bots into groups:

Flag Carrier One bot with the task to capture the enemy flag and return it

Roamer Two bots with the task to eliminate the enemy team

Flag Defender One bot with the task to defend their own flag

We chose this design, because of various reasons. One of them is when every bot has the same task, to capture the enemy flag, they would get in each others way. Also this would be very inefficient because no one would defend their own flag. The bots should also have the ability to pickup weapons, armor and health. This will give a large boost, and the chances of winning will increase. Another thing which is important, is to return the our own flag if it is dropped by the enemy. Otherwise the bots can not score any points. There also should be strategy's to pickup things. You do not want to grab health when your life is full for example. At last the bot should switch weapons based on the distance between him and the opponent. If they are close to each other, he should switch to a weapon with a high close-range damage vice versa.

2.2.3 Extended Functionality

Functionalities that we could have are:

- Another agent, one who has an overview of the game and tells the other bots what to do
- When the one of Roamers or the Flag capturer grabs a flag, he will be followed by the other, who will defend him against enemy bots.
- An agent, who has only one task, to control the respawn of the bots. Sometimes, the bots get stuck, and they are idle for some time. The agent should handle this, wasting no time.

The idea to make a controlling agent is a good one. This would make the implementation more accessible, but on the other hand, the program would run very slow. The controlling agent has to send a lot of messages to the other agents, and this would cost a lot of time. There would be a significant delay in executing tasks, which would decrease our chance at winning the game. So we will not have this functionality.

Chapter 3

Analysis of the UT environment

The ten scenarios are:

1. The first scenario is when the Flag capturer captures the enemy flag. His goal is to grab the enemy flag, and when he percepts that he is holding the flag, his goal is to bring it back to his base. If he does not encounter an enemy, he heads straight to the base. When he does encounter an enemy, his priority is killing the enemy. If he succeeds, he continues delivering the flag. Eventually he will be defended by the roamers and Defender, so he can easily run back to his base.

2. The second scenario is when the Flag Defender encounters an enemy. At first, he roams around his own base. If he is stuck, he automatically respawns. When he sees an enemy, his priority is to kill him. When the Defender is hit, he will go grab health. If he is killed by the enemy, the Defender respawns and will roam around his base again. When he sees the same enemy as before, he will again try to kill him. But if he can not see the enemy anymore, he will not try to pursue him, bot stay around the base. We chose this, because we did not want that the Defender would leave his position, otherwise we would be very vulnerable. The enemy who got away, should then be taken care by the roamers and perhaps the Flag Capturer.

3. The third scenario is when the roamers see an enemy. If they see an enemy, their priority is to kill them. They will pursue the enemy trying to kill them. When the flag capturer has grabbed the enemy flag and runs back to his base, and he is pursued by an enemy, the roamers automatically set their goals to kill them. The fact that the enemy is pursuing the Capturer, is not important. This is what makes the roamers so lethal.

4. The fourth scenario is when one of the roamers is hit. When the enemy hits one of the roamers, and their health is dropped below 100, he will adopt a new goal to leave the battlefield and grab some health. This will only happen if he sees a health pack. When his health is 100 again, he goes back to defeat the enemy if he sees one. There is also a big health, which boosts it to 180. If the roamer sees one, he will adopt the goal to grab it.

5. The fifth scenario is when the roamers are low on ammo, or when they see a new weapon, or armor. If they percept that they are low on ammo, the new goal is to grab the ammo. When they have grabbed it, they will continue with what they were doing. If the bot sees a weapon, he will first check if he already has the weapon. If he does not have, his goal will be to pick the weapon up. This also counts for armor, he will first check if he already has this specific armor, if not, he will try to grab it.

6. The sixth scenario is when the roamers end up capturing the flag. If the roamers does not have any goals, he would be inactive. Instead of being inactive we decided that when he reaches this point, he should go to the enemy base and try to grab the flag. Once he captures the flag, he basically becomes a Flag Capturer, and he will try to deliver it back to his own

base. If he encounters an enemy while holding the flag, he will try to defeat him, and when that happens, he will continue delivering the flag.

7. The seventh scenario is when the Flag Capturer sees a weapon. He percepts that there is a weapon, then he looks at his beliefs and checks if he does not own that weapon already. If he owns it, he will continue trying to capture the enemy flag. If he does not owns it, he will adopt the goal to pickup the weapon. After the grabbing it he will continue trying to capture the enemy flag.

Chapter 4

Design of an Agent System

4.1 System Structure

Our system has three distinct roles for our four agents:

1. The first being the *Carrier*, designed to be optimal in retrieving and capturing the enemy flag. Once the flag has been captured, the *Carrier* will often be defended by one of the *Roamers*, for added security.

2. The second is the *Defender*, designed to defend our flag and home base. If our flag has been captured, he should chase the enemy flag carrier and re-capture the flag.

3. The last role is the *Roamer*, of which there are two agents running in a match. They are designed to generally walk over the entire map, getting a lot of weapons, health and armour and disrupting the enemy team. Additionally, they can capture the flag if they have nothing to collect or go after the enemy flag carrier if our flag is stolen.

4.1.1 Justification

4.2 Strategy of the Agent Team

4.2.1 Justification

4.3 Agent Team Validation

Chapter 5

Testing Approach and Results

Chapter 6

Conclusions and Recommendations

Part II

Appendix

Appendix A: Ontology

armor(w, x, y, z)	Shows which types of armor the bot currently possesses (w is helmet, x is vest, y is pants, z is belt).
at(x)	is true if the bot is at x.
bringFlag(x)	x is an UnrealID to which the flag should be brought.
currentWeapon(x, y)	x is the current weapon the robot has equiped, y is the current firing mode.
flag(x, y, z)	x is the team that currently possesses the flag, y is the HolderUnrealID of the flag and z is the position of the flag.
flagState(x, y)	x is the team, y is the current status regarding the flag.
getFlag(x)	x is an HolderUnrealID from which the flag should be gotten.
goTo(x)	x is an UnrealID to which the robot should go.
item(w, x, y, z)	w is the UnrealID of the item, x is its label, y the ItemType and z is the position.
location(x, y, z)	x, y and z are the coordinates of this location.
look(x)	x is UnrealID to which the bot is looking.
navigation(x, y)	y shows the location to which we try to navigate, x shows the current status (stuck, noPath, reached).
navigation(x, y, z)	x is the current location of the both, y is the current rotation and z is the current velocity of the bot.
path(w, x, y, z)	shows a path is available from w to x with length y. z is a list of locations.
pickup(x)	x is the UnrealID of the item we just picked up.
powerup(x, y)	x shows which powerups the bot currently has equipped, y is the remaining time.
prefer(x)	a list of weapons, sorted on preference (most preferred weapon first).
score(x, y, z)	the current score, x is the number of kills, y is the number of deaths and z is the number of suicides.
shoot(x)	x is the TargetLabel on which the bot will (try to) fire.
status(x, y)	x is the health of the bot, y is the amount of armor it still possesses.
wantArmor(x)	shows which armor the bot still wants (x is an armor identifier).

Appendix B: Progress report

Appendix C: Account of hours