

# DEVELOPING AN AI LEARNING AGENT FOR A 2D FIGHTING GAME USING MACHINE LEARNING

Final Year Dissertation

Computer Systems BSc (Hons)



# Abstract

Video games have had the same general design since the beginning, players interacting with various characters in some way. All characters behave a certain way depending on the video game, this makes it more interesting, but almost always, this behavior is scripted. Scripted behavior makes the characters predictable. In a short amount of time the players find the pattern of the characters, which in turn makes the game unenjoyable. Instead of having scripted behavior, if characters can keep the player's sense of enjoyment by maintaining the player's "Flow" as well as learn and improve with the player throughout the video game, it could add new dynamics to future video games.

Machine Learning techniques can and have been applied to video games, but they are not often used, If implemented correctly, machine learning can create video games that can learn as the player learns. For instance, this could replace difficulty caps since the video game NPCs could just learn to play, from the player, the game would get harder as the player gets better at the game. This is just one of the many applications, other than that machine learning could add more depth to a video game which would make it even more immersive for the player.

This project focuses on implementing machine learning techniques in a simple fighting game that is going to be developed specifically for this project. The game would have generic actions such as, move forward, backward, jump, punch, kick, block. The machine would learn to play from the player, based on trial and error.

# Contents

1. Introduction .....	8
1.1. Aims .....	8
1.2. Objectives .....	8
2. Background.....	10
2.1. AI .....	10
2.1.1. What is intelligence? .....	10
2.1.2. What is AI? .....	11
2.1.3. Intelligent Agents .....	13
2.2. Machine Learning .....	20
2.2.1. What is Machine Learning?.....	20
2.2.2. Difference Between Machine Learning and Artificial Intelligence .....	20
2.2.3. Machine learning techniques and algorithms .....	21
2.2.4. Applications of Machine Learning .....	24
2.3. Video Games.....	25
2.3.1 Flow Theory .....	25
2.3.2. Uncertainty in video games .....	27
2.3.3. Video games that learn .....	27
2.3.4. Machine learning in Fighting games.....	28
3. Project Design .....	29
3.1. System structure.....	29

3.2.	Development stages .....	29
4.	Implementation .....	30
4.1.	Tools and environments.....	30
4.1.1.	TensorFlow .....	30
4.1.2.	TensorBoard .....	30
4.1.3.	Unity Engine .....	30
4.1.4.	Unity ML Agents .....	31
4.1.5.	Nvidia CUDA Toolkit .....	31
4.1.6.	Nvidia cuDNN .....	31
4.1.7.	Visual Studio.....	31
4.1.8.	Anaconda .....	32
4.1.9.	Python .....	32
4.1.10	C#.....	32
4.2.	Machine Learning Environment.....	32
4.2.1.	Environment Structure .....	32
4.2.2.	Agent .....	33
4.2.3.	Brain .....	36
4.2.4.	Academy.....	37
4.3.	Game.....	37
4.4.	Training the agent.....	38
5.	Evaluation .....	40

5.1.	Evaluation Methodology .....	40
5.2.	Evaluation cases.....	40
5.2.1.	Case 1.....	40
5.2.2.	Case 2.....	43
5.2.3.	Case 3.....	45
5.3.	Outcome .....	48
5.3.1.	Case Comparison.....	48
5.3.2.	Testing the cases .....	49
5.3.3.	Justifying test results .....	49
6.	Conclusion .....	51
6.1.	Project Summary .....	51
6.2.	Project Management.....	51
6.3.	Project Outcomes .....	51
6.4.	Professional, Legal and Ethical Issues.....	52
6.4.1.	Professional Issues.....	52
6.4.2.	Legal Issues .....	53
6.4.3.	Ethical Issues.....	53
6.5.	Reflection and Future Work .....	53
7.	References .....	55
8.	Appendix.....	59
	A1 - Functional Requirements.....	59

A2 - Risk Analysis .....60

# 1. Introduction

Since the first video game, Pong, was created the main concept of video games has not changed. Even though the video game industry has progressed to a very high standard the elements that make up the game stay the same. Players play, with or against a NPC (Non-Player Character) that has a scripted behavior. Although this adds immersion to the game, it also makes the game somewhat unenjoyable if the player wants to play the second time. The scripted behavior, affects the replay ability of the game. This issue can be solved by, instead of scripting behaviors, implementing Machine Learning agents to the existing NPCs in such a way, so that they start learning from the environment and the player. This would also keep the NPC's skill level similar to the player's skill level at all times since the NPC would be learning from the player, the NPC will get good as the player gets good resulting in good maintained flow which in turn would result in an increased replay ability of the game.

## 1.1. Aims

This project aims to develop an AI agent using Machine Learning techniques, that is able to learn from the player's behavior in a 2D fighting game which will be developed specifically for this study.

The Game should function like a standard fighting game with a basic move set, such as kick, punch, block, and the agent should be capable of learning the basic move set from the player and adapt its fighting style over time depending on the players actions.

## 1.2. Objectives

- Develop an AI agent that learns the core elements of the game from the player.
- Develop a 2D fighting game for the agent.
- The agent should be, to an extent, unpredictable after it has learned the core elements.



- The Implementation and design of the Agent should be flexible to adapt to any changes that might be made to the game.
- The learning agent should be capable of running autonomously until stopped.

## 2. Background

This section contains all relevant information needed to understand the project properly.

### 2.1. AI

Humans have always been trying to understand themselves and it is due to our fascination with intelligence that we like to think of ourselves as an intelligent species. Humans are so intrigued by their intelligence that they have tried to replicate it, thus giving birth to the idea of Artificial Intelligence. The term Artificial Intelligence or AI has been roughly defined by one of the major pioneers, John McCarthy, in 1955, as “The goal of AI is to develop machines that behave as though they were intelligent” <sup>[1]</sup>. This is a very basic definition, to accurately understand what the term artificial intelligence is, we first need to understand what intelligence is on its own.

#### 2.1.1. What is intelligence?

There is no set definition of intelligence. In general terms intelligence is not successfully performing complex tasks. Intelligence is adapting one’s behavior in such a way, so that it fits to any circumstance that might arise <sup>[2]</sup>.

A simple human behavior can be considered intelligent whereas quite complex behaviors noticed in insects can be considered unintelligent. Consider the behavior of some insects, when the insect performs an action it follows a procedure, no matter how many times, the insect will follow the same procedure each time it performs that specific action <sup>[2]</sup>. A good example is the female digger wasp. Each time the female digger wasp brings back food, it places the food outside the burrow and goes inside to check if the burrow is free of intruders then if the burrow is clear, it takes the food inside. The unintelligence of the wasp’s behavior can be seen if the food that it placed outside, is moved when it goes in to check the burrow. The wasp goes through the whole procedure again, it places the food outside once again, checks the burrow

and comes back out. The food can be moved more than forty times and the wasp would still carry out the same procedure each time, its behavior is not considered intelligent <sup>[2]</sup>.

Intelligence is not a single ability or a cognitive process, it is a set of different cognitive processes that work together. The processes are:

- The ability to reason
- The ability to learn
- The ability to solve problems
- Perception
- The ability to understand language

Similarly, since artificial intelligence and intelligence are correlated, the research in artificial intelligence mainly focuses on the same cognitive processes mentioned above <sup>[2]</sup>.

### **2.1.2. What is AI?**

Just like intelligence there is no set definition for Artificial Intelligence that is accepted universally. Artificial Intelligence is, in simple words, intelligent behavior exhibited by machines. The term artificial intelligence is mainly used for a machine that mimics the cognitive processes mentioned in the last section (What is Intelligence?). The advances in computational performance and the accuracy of algorithms, makes it challenging to define a distinction between what constitutes as Artificial Intelligence and what does not. The boundaries of AI are not set, they tend to change over time <sup>[3]</sup>. There are so many problems and solutions revolving around AI that help increase advances in the field, due to this there is no set definition of AI.

There are three main types of Artificial Intelligence:

- Strong AI
- Narrow AI (Weak AI)

- Broad AI (General AI)

### **Strong AI**

Strong AI's are aimed at accurately simulating the reasoning ability of a human. This type of AI can not only be used to build systems that can think for themselves but also help understand how the human cognitive processes work. The field of AI is still not advanced enough therefore strong AI's are not yet built, it is merely an idea for now <sup>[4]</sup>.

### **Narrow AI (Weak AI)**

Weak AI's are aimed to just make a system work, they are designed for a specific task. This type of AI can also be used to build systems that can think for themselves, but the results would not help in understanding how humans think. A prime example of this type of AI was IBM's Deep Blue, a chess playing machine that beat the chess world champion. The machine did not play like a human therefore it did not help understand the cognitive processes of a human <sup>[5]</sup>.

The advantage of a Narrow AI is that it is focused on one specific task, making it extremely good at the task it was designed for. The disadvantage is that it is completely useless for anything other than the task it was designed for.

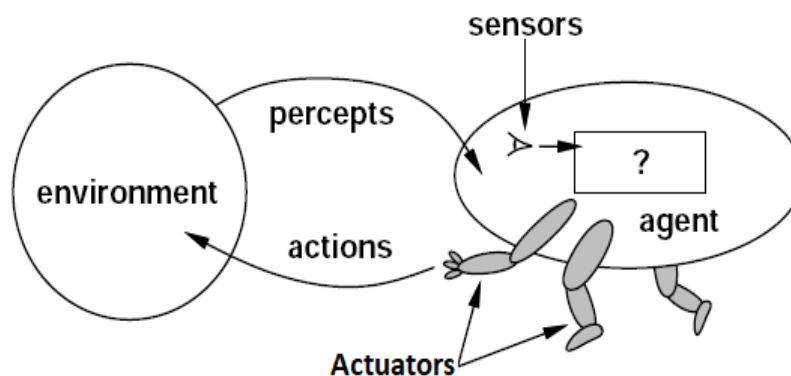
### **Broad AI (General AI)**

Broad AI's are mainly aimed to make a system that is based around the ability to reason. This type of AI is a machine that can use and apply intelligence to any situation or problem it is given. Using its reasoning ability, it can overcome any problem not just one specific problem <sup>[4]</sup>. As machines constantly advance and the tasks they do become more complex the need for general AI increases.

Artificial Intelligence has been an attraction for researchers for a long time, the term was coined almost 60 years ago, in 1956 and the research on AI in the computer science field was defined as the study of Intelligent Agents.

### 2.1.3. Intelligent Agents

An intelligent agent is a system that is autonomous and acts on an environment with the help of sensors and actuators so that it is able to achieve the goals it was designed for.



**Figure 1 – A general intelligent agent**

As shown in Fig.1, This system uses sensors to observe the environment it is in and then acts upon its environment with actuators which helps the system achieve its goals. Being autonomous in this case doesn't necessarily mean the system can think for itself, it means that the system can act on its environment without any direct human intervention and that it can control its own actions as well as control its own internal state <sup>[6]</sup>. A simple example of an intelligent agent would be a smart thermostat. The thermostat uses its sensors to percept how hot or cold the climate is and then changes the temperature using actuators, all of this is done without a human stepping in.

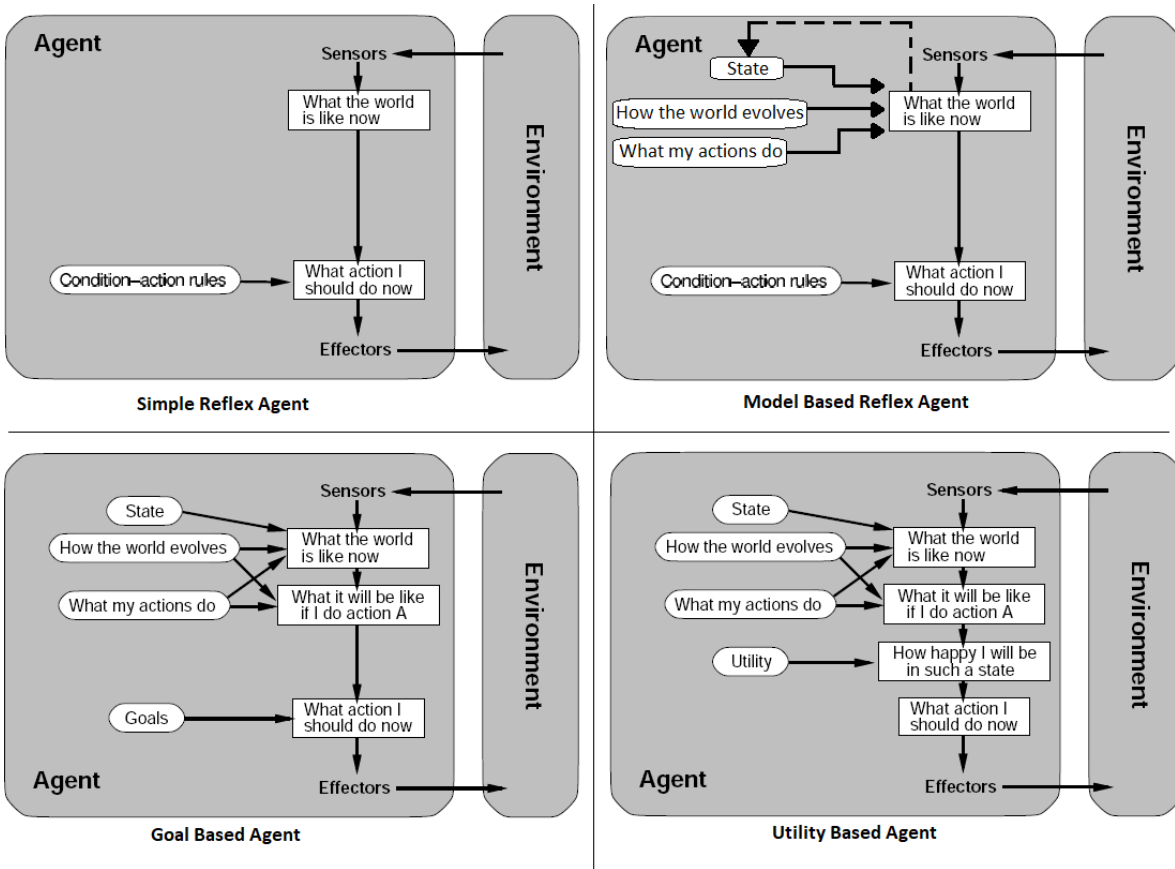
Intelligent agents branch into many classifications, based on the type of environment they are placed in, the agent's capabilities, the agent's properties, what goals is the agent intended to

achieve or how they act upon the environment. An agent does not necessarily have to fit into a specific category or classification it can fit into more than one classifications <sup>[8]</sup>.

According to Norvig and Russel <sup>[7]</sup>, intelligent agents can be grouped into five classes that is based on how intelligent and capable an agent is. The classes are ordered by the increase in complexity and intelligence of the agent, they are:

- Simple reflex agents
- Model based reflex agents
- Goal based agents
- Utility based agents
- Learning agents

As shown in Figure 2, the first four classes have a similar structure, and expand upon each other based on the order of increasing complexity and intelligence. From the first four classes each class is an iteration of the previous one, with respect to the list above, adding and/or improving upon features from the other.



**Figure 2 – The first 4 types of intelligent agents and their processes**

### Simple reflex agents

Generally, intelligent agents collect more than one perceptual input and then base their actions on those collected inputs but, a simple reflex agent bases all its actions only on one input, that is the current perspective of the environment that it has. It does not base any of its actions on the past perceptions of the environment <sup>[9]</sup>. Simple reflex agents are useful when a quick automatic response is required, a similar example of this type of response can be found in humans, when touching something very hot the human brain instantly pulls the hand away without thinking of any other action. These agents are based on the *condition-action* rule: If a condition is met then do an action <sup>[9]</sup>.

Simple reflex agents are frequently used in video game NPCs because they are easy and fast to implement. An award-winning racing game called “Obliteracers” uses simple reflex agents for its NPC racers. The NPCs collect relevant inputs about the racer state, racer position, distances, and directions of other racers, from the race track then they decide on the best action to take using *condition-action* rules and instantly react to the condition it chose.

Although this is a good choice for a racing game this type of agent does not learn from its environment it simply acts upon it therefore it would not be a good choice for this project.

### **Model based reflex agents**

Unlike simple reflex agents, a model based reflex agent bases its actions on all previous perceptions of the environment that it has. These types of agents store a type of model of the environment’s current state, thus the name Model based reflex agents. The stored model is a collection of all states that the agent has previously observed, and it is updated each time the agent observes the environment <sup>[10]</sup>.

An example of this type of agent can be seen in video games with patrolling enemies, when the player gets noticed by an enemy, the guard starts to chase the player and stores that state, if the player hides the guard tries to look for the player until some time limit has reached. This behavior of the guard is based on all the stored states of the environment. This would not be possible if a simple reflex agent is used, since they base their actions only on the current state, as soon as the guard loses sight of the player the guard will immediately stop chasing <sup>[10]</sup>.

Again, these types of agents are a bit more intelligent than simple reflex agents, but it would still not be a good choice for this project since they just store the environment state and do not really learn it.



## **Goal based agents**

Goal based agents have similar capabilities as model based reflex agents, the only difference is that Goal based agents have set goals that they actively try to pursue. These types of agents use search and planning, this is because they consider multiple scenarios before taking any action on the environment to see which action will result in a goal achieved <sup>[11]</sup>.

Taking the same example of the video game with patrolling guards, if goal based agents are used then the guards can have goals, for example capturing the player. Now the guard will not just blindly follow condition-action rules but rather consider different scenarios to capture the player <sup>[10]</sup>.

Goal based agents may seem like they learn from their environment but in reality, they do not learn, they just consider different paths leading to a predefined goal.

## **Utility based agents**

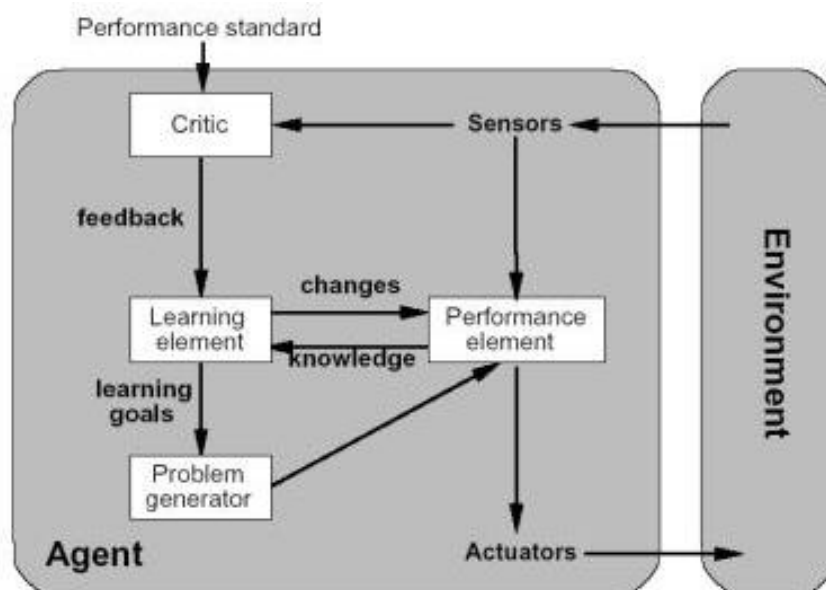
Utility based agents expand more on the capabilities of goal based agents. Goal based agents only consider scenarios leading to goals, they cannot rate goals that are more beneficial at that moment, from a set of goals. Utility based agents on the other hand are able to rate all scenarios in the environment using a utility function, the rating is based on how well the agent achieves a goal with regards to how good an outcome of a scenario is (refer to Figure 2) <sup>[11]</sup>.

A tactical First-Person Shooter video game, Killzone 2 has been praised to have the best designed AI, its developer Guerilla Games used utility based agents for its AI so that it could make good tactical decisions <sup>[12]</sup>.

Although Utility based agents can create good AI and make good decisions on their own they still do not learn. This can be proven if a utility based agent is initially placed in an unknown environment, it will not be able to make the right decisions until it stores the new world state.

## Learning agents

Lastly, as the environments get large and the number of tasks increase, the number of condition-action rules to pre-define will also drastically increase, managing all the rules can be difficult. This problem can be avoided if an agent is able to learn new actions while it is doing the task it was designed to do. This agent is used in Machine learning. Learning agents are somewhat different from the first four agents in a sense that the first four agents could not operate in initially unknown environments whereas, learning agents can operate in them. The main goal of these agents is to improve performance. Learning agents operate based on four parts working in conjunction, as shown in Figure 3 <sup>[9]</sup>.



**Figure 3 – A learning agent (machine learning agent) and how it processes**

The 4 parts are:

- The Learning element
- The Performance element
- The Critic
- The Problem generator

The Learning element helps the agent improve by making changes to the knowledge components in the agent. Learning can be achieved by observing pairs of successful environment states that the agent has stored, this information can help the agent learn how the environment evolves. For utility based learning agents, a performance standard is required to tell the critic if the agent's action has a good or bad effect on the environment <sup>[9]</sup>.

The Performance element selects external actions, this is considered to be the first four agents (figure 2).

The Critic provides feedback to the learning agent, on the agents own performance. The agent then decides how the performance element needs to change so that it can improve.

The Problem generator, as the name implies, generates different problems for the agent so that it can experience new situations and then learn from them. The problem generator is there to ensure that the agent continues to learn <sup>[9]</sup>.

An example of learning agents used in the video game industry is the popular racing game, Forza Motorsports. In Forza the NPC racers do not use standard AI techniques, they use machine learning (learning agents). These NPCs are called "Drivatars", Players can train them by driving on a set of tracks that are designed to contain challenges. The game then stores the path taken on each of the tracks and uses that data to produce an NPC that drives like the player <sup>[13]</sup>.

Machine learning is going to be implemented in this project in a similar way, where the player will have to play against an NPC to help it learn.

## **2.2. Machine Learning**

### **2.2.1. What is Machine Learning?**

The origin of Machine Learning dates back to 1959, when a pioneer in the field of artificial intelligence and computer games, Arthur Samuel, coined the term “Machine learning”. Machine learning, as the name implies, gives machines the ability to learn from experience without the need for specific programming <sup>[14]</sup>. Its main goal is to program machines in such a way that they can learn to solve any given problem using data collected from past experiences <sup>[16]</sup>. Machine learning works based on algorithms that collect data to learn from the environment and then make predictions or decisions based on that collected data, this overcomes the need to follow strict condition-action rules.

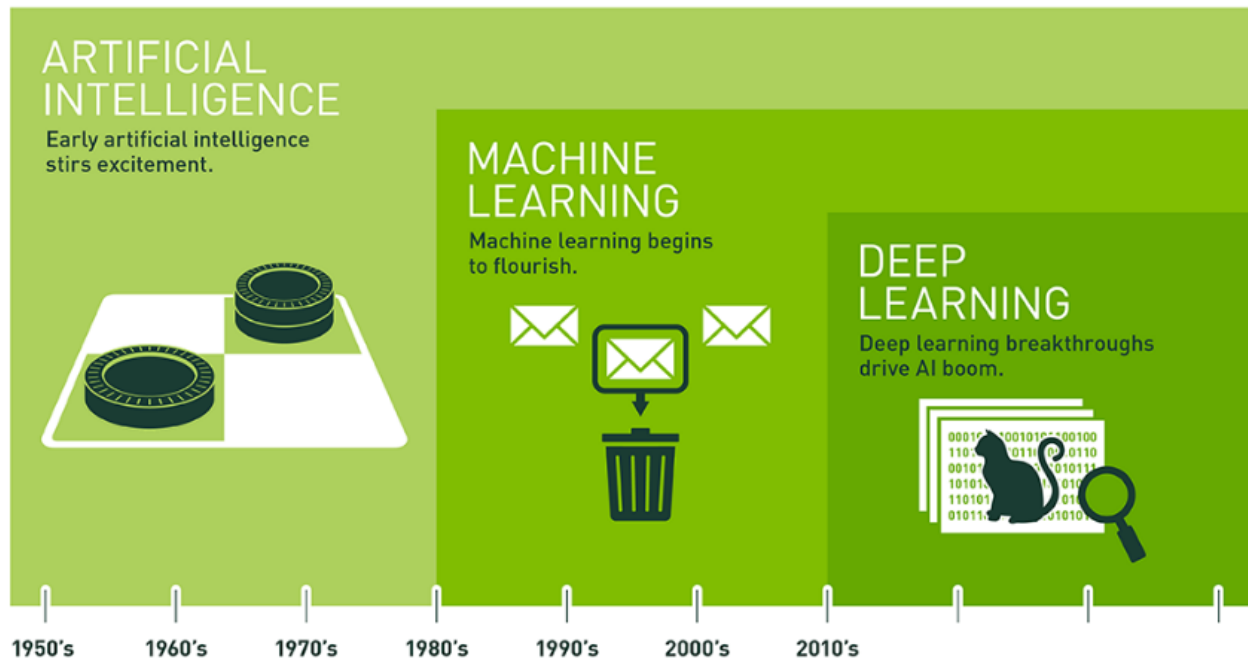
Over the past years machine learning has produced many things such as, face recognition, self-driving cars, even video game NPCs like in Forza <sup>[13]</sup>, but these things are not commonly correlated with machine learning but rather AI. While machine learning and AI are closely related, they are different from each other.

### **2.2.2. Difference Between Machine Learning and Artificial Intelligence**

Artificial Intelligence is a branch of computer science that is based on building machines that are capable of behaving intelligently, this behavior can be programmed. Machine learning on the other hand, gives machines the ability to learn from experience without the need for specific programming <sup>[17]</sup>.

The easiest way to understand the relationship between AI and machine learning is to think of it as a parent-child scenario, AI is the parent, it is the idea that came up first and Machine learning is the child, which came later. As shown by Figure 4, AI is the first then around 1980 Machine learning begins to popularize and then finally around 2010 Deep learning, a subfield of Machine

learning is introduced <sup>[17]</sup>.



**Figure 4 – AI and its subfields with the dates they were popularized in**

Just as AI has different techniques, machine learning can also be implemented using various techniques and algorithms, Deep learning being one of the techniques of implementing machine learning.

### **2.2.3. Machine learning techniques and algorithms**

Machine Learning algorithms are often divided into 4 main categories. They are:

- Supervised Learning
- Unsupervised Learning
- Semi-supervised Learning
- Reinforcement Learning

## Supervised Learning

The basic idea of supervised learning is that the machine is inputted with training data. That data provides examples of scenarios and specifies an outcome, called a label, for each of the scenarios. The machine can use this training data to build a model which can then predict the output of **new data** that the machine is given based on the past examples provided by the **training data**. The main goal of a supervised learning algorithm is to successfully build a model that can predict the output of new scenarios and cases that it comes across <sup>[18]</sup>.

Some of the common algorithms that use Supervised learning are:

- Neural Networks
- Decision trees
- Linear Regression
- Nearest Neighbor

An example of supervised learning applied to a video game is a program called “Marl/O”. This program is made of neural networks which plays the classic video game “Super Mario” with ease. The program was left running and eventually it learned to play the game from scratch and complete a level <sup>[19]</sup>.

## Unsupervised Learning

Unsupervised learning is also inputted with training data, but this time the training data only consists examples of scenarios and not the labels. The machine builds a model to predict the output of new data. This can be done by comparing and grouping similar training data which is called clustering <sup>[18]</sup>.

Some of the common algorithms that use unsupervised learning are:

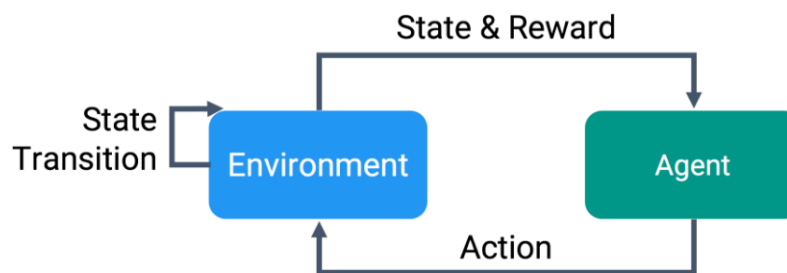
- K-means clustering
- Association rules

## Semi-Supervised Learning

In semi-supervised learning the machine is provided with both, training data that has labels and training data that does not. Semi-supervised learning is a mix of supervised and unsupervised learning and therefore produces the model in the same way the other two do <sup>[18]</sup>. This category of learning can be used when there is some labeled data as well as unlabeled data for a scenario.

## Reinforcement Learning

Reinforcement learning is based on a reward system. The machine constantly interacts with its environment and receives a reward for each of its actions. The main goal of the machine is to increase its reward or decrease the risk over a sequence of actions and iterations with the environment <sup>[18]</sup>. As shown by Figure 5, reinforcement learning algorithms keep learning from experience of the environment until they explore all possible states.



**Figure 5 – A representation of how reinforcement learning works** <sup>[20]</sup>

Some of the common algorithms that use reinforcement learning are:

- Q-Learning
- Temporal Difference (TD)
- Deep Adversarial Networks

Reinforcement learning has helped build machines that have beaten world champions. Machines like Deep Blue, the chess playing machine and AlphaGO, the GO playing machine, would not be possible without reinforcement learning <sup>[18]</sup>.

This project will mostly lean more towards using a combination of Supervised learning, neural networks and Reinforcement learning, Deep learning.

#### **2.2.4. Applications of Machine Learning**

As the field of machine learning evolves its real-world applications are also getting very diverse. Compared to a few years ago machine learning now is being used almost everywhere. These are some of the applications of machine learning being used now:

- Cognitive services
  - Text to speech
  - Speech to text
  - Language translation
  - Visual recognition and Facial recognition
- Medical
  - Diagnosing cancer: Googles Deep learning AI is being used to diagnose cancer. It resulted with 89% accurate diagnosis.
- Business
  - Predict the trade prices
- Entertainment
  - Used to make music: a company called “Jukedek” uses neural networks to compose music.
  - Used in Video Games: One of the popular examples of this is the game “Forza Motorsport” as discussed earlier.



## **2.3. Video Games**

Due to the video game industry being very strict and unforgiving, companies do not try new things due to fear of falling within the industry instead they just stick to standard AI, that is one of the main reasons why not many commercial video games have used machine learning, but that is changing. As computers become more powerful and cheaper, the demand for the video game industry has increased as well. Statistics show that, in 2004 the sales of entertainment software reached \$25.4 billion worldwide and it is increasing <sup>[21]</sup>.

Due to the constant increase in video game demand, video game companies are forced to bring something new to games that is if they do not want to fall in the industry. Therefore, these companies don't just have to focus on making a good video game but also on the playability of the game. They must make sure the game is enjoyable for all the players, companies do this by taking into account a psychological concept known as Flow theory.

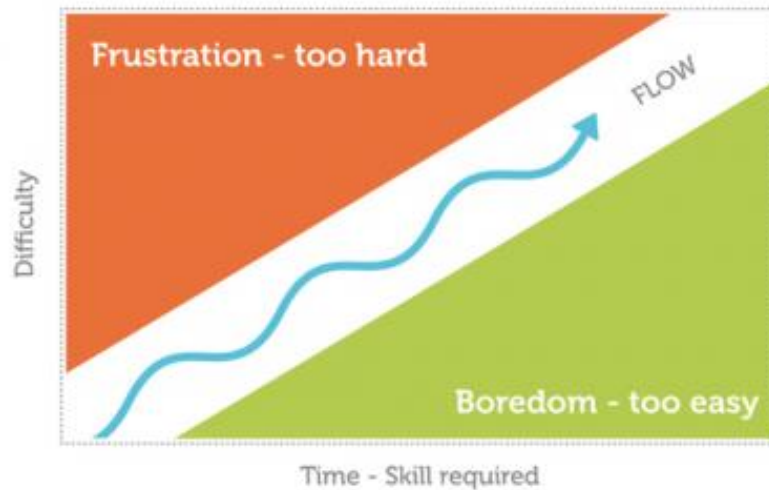
### **2.3.1 Flow Theory**

The concept of Flow Theory is the works of a psychologist Mihaly Csikszentmihalyi who studied something he called the mental state of "flow", it is also sometimes called as being "in the zone". This state is when one is energized and has complete focus in an activity along with a high level of enjoyment, for video game players this can be when they are fully immersed in the game and lose track of time. According to Csikszentmihalyi, the state of flow is achieved when a task or challenge is in balance with one's skill level, if they are not in balance the task can range from being boring to frustrating <sup>[22]</sup>. Flow theory has been applied to many fields to improve human experiences.

Video games is one of the major fields that gives importance to applying Flow Theory, this is due to the fact that video games are meant to be enjoyable. Games must be made in such a way so that they can cater to players of all skill levels. For instance, if a player's skill level is low

and the video game difficulty is high the game would be frustrating for that player. On the other hand, if a player's skill is high and the video game difficulty is low, the game would be very easy thus being very boring for that player.

Figure 6 shows a Flow Model, that shows flow in terms of difficulty and time-skill required.



**Figure 6 – Flow Model for games** <sup>[23]</sup>

A machine learning NPC can easily maintain a player's flow, the reason being that the machine would learn from the player and therefore, it would stay close to the players skill level. As the player becomes better at the game, the NPC will also get better. The difficulty will increase at a same rate as the skill and therefore, result in the player achieving flow.

Some video games design NPCs in such a way where they follow the same pattern each time. This is good for new players since at that point the new player's skill is not that high and is somewhat balanced with the games difficulty, but over time when the pattern gets repetitive and the player's skill increases more than the game difficulty, the game becomes very boring for the player. Successful video game companies maintain player flow and at the same time avoid being too repetitive, they do this by adding a bit of uncertainty to the games.

### **2.3.2. Uncertainty in video games**

Uncertainty is a primary characteristic of a good video game. At each point in a good game that requires skill, the player is at risk of dying, getting defeated or losing, without this uncertainty the game would not be enjoyable. If a game becomes too predictable the players can easily anticipate what is going to happen which in turn would make the player bored of the game <sup>[24]</sup>. For instance, if a person played the same video game twice, he would enjoy more the first time compared to the second time.

Uncertainty is necessary to make an enjoyable game, consider a game in which the player is very good and can effortlessly win the game, the player would not find that game enjoyable because it is too easy. The player is certain that he is not going to lose, there is no challenge for him thus making the game boring <sup>[24]</sup>.

A good video game NPC that uses machine learning should not only learn from the player, but it should also be unpredictable, it should have a little uncertainty so that playing against the NPC is enjoyable for the player.

### **2.3.3. Video games that learn**

A case study on video games that use machine learning called “Creating Intelligent Agents in Games”, discusses a game called “NERO”. The NERO Game puts the player in the role of a military trainer who teaches a team of NPCs by making a program for them. The NPCs are simulated robots and the players goal is to get them ready for combat. The NPCs start with no skill, only the ability to learn. The player designs different training exercises and goals with increasing difficulty to prepare the NPCs for combat. Increasing the difficulty for each exercise results in the NPCs learning the basics and then gradually learning the advanced strategies <sup>[21]</sup>.

This game uses a machine learning technique called “neuroevolution” or in simple terms, evolution of neural networks, which is an adaptation of another machine learning technique

called Neural network. Neural Network is a machine learning technique that is modeled based on the natural way things evolve also known as natural evolution <sup>[21]</sup>.

#### **2.3.4. Machine learning in Fighting games**

This project is mainly based on how machine learning can be implemented in fighting games, therefore an understanding of fighting games is crucial. Fighting games, as the name implies, usually involve characters who fight, in most cases one to one or hand to hand combat. Some of the famous fighting games are, Tekken, Mortal Kombat, Street Fighter, Injustice and Soulcalibur <sup>[25]</sup>.

A study named “Real-Time Imitation Based Learning for Commercial Fighting Games” talks about how Reinforcement learning was applied to a fighting game which gave interesting results. The study also talks about another machine learning approach called imitation learning <sup>[26]</sup>. This technique was used in one of the biggest fighting game franchises, Tekken. Imitation learning was used in Tekken 5 Dark Resurrection, when a player played the game, the game would produce an imitation data file for that specific player and then other players can obtain that file to play with an imitative Artificial Intelligence. This technique was later known as a Ghost AI system. This system recorded various actions of the player in different situations, the system then generated an AI that would then perform like the player it imitated. Players were able to play with the AI as if playing with the person who trained the AI <sup>[26]</sup>.

This project aims to achieve the same but instead of imitating the player the agent learns from its environment and the player from scratch.

## **3. Project Design**

### **3.1. System structure**

In this project, the AI learning agent that is developed will learn to play a 2D fighting game. It will learn basic actions that will drive the agent. The game is specifically developed for the agent, to avoid any compatibility or implementation issues that may arise otherwise. Both the game and the agent are developed in Unity. Although the game can be developed without any external libraries, the agent on the other hand uses Unity's ML-Agents SDK along with a Machine learning library, in this case TensorFlow.

### **3.2. Development stages**

This project is split into two main stages:

- The development of a fighting game
- The development of a learning agent

During the initial phase of the project the game concept and plan was created. Once the concept was finalized and the decisions were made a bare bones version of the game was developed, this version only included the main elements of the game such as, the player models, the actions and the stage. After this was done the agent implementation became the main focus of this project. The agent was developed in iterations with each iteration expanding on the last. The agent was then implemented in the game, trained, and tested. After the agent worked properly and was trained the development of the game was continued and finalized.

## 4. Implementation

### 4.1. Tools and environments

Some of the tools used for the development of this project are relatively new therefore they will be explained in detail for the purpose of better understanding the implementation. The tools and environments used are the following:

#### 4.1.1. TensorFlow

TensorFlow is an open source library developed by Google that is used for high performance and complex mathematical computations. It can be used on numerous platforms such as GPUs, CPUs, TPUs (Tensor Processing Units), and can now be also used on mobile devices <sup>[27]</sup>.

TensorFlow strongly supports machine learning and therefore, is being used in this project. This project uses TensorFlow-GPU which helps in training the learning agent, and TensorFlowSharp which is just the normal TensorFlow library which is made to work with the C# programming language.

#### 4.1.2. TensorBoard

TensorBoard is a visualization toolkit for TensorFlow. Computations in TensorFlow can be very confusing and complex and TensorBoard makes it easy to understand these computations by laying out relevant data in the form of graphs and charts, in the case of this project it is data such as the learning-rate/time, cumulative-rewards/time, entropy/time. **TensorBoard** is ran during the training process, it reads event files that contain summary data that are generated while **TensorFlow** is running <sup>[28]</sup>.

#### 4.1.3. Unity Engine

Unity is a game engine developed by Unity Technologies, mainly used to develop video games for computers and consoles. It supports 2D and 3D graphics and scripting using C#. Although there were other options to choose from, Unity was mainly chosen because of its recently

released ML-Agents SDK built for machine learning. Other than the SDK, Unity also has a vast amount of material online compared to other options. Due to these reasons the main scene and the game for this project is developed using Unity.

#### **4.1.4. Unity ML Agents**

As mentioned above Unity has released an SDK called Unity Machine Learning Agents. The ML agents can be used to train intelligent agents using Evolutionary Strategies, Deep Reinforcement Learning and other machine learning techniques using a Machine Learning Library like TensorFlow and a Python API <sup>[20]</sup>.

#### **4.1.5. Nvidia CUDA Toolkit**

CUDA short for Compute Unified Device Architecture, is a programming model and a parallel computing platform that makes it possible for GPUs to perform numerical calculations. The Toolkit provides an environment to create high performance applications that are accelerated by GPUs <sup>[29]</sup>. The CUDA Toolkit was chosen for the development of this project because it includes various GPU accelerated libraries and is also compatible with TensorFlow-GPU.

#### **4.1.6. Nvidia cuDNN**

The cuDNN (cuda Deep Neural Network) is a GPU accelerated library of CUDA for deep neural networks <sup>[30]</sup>. This library helps in accelerating frameworks such as TensorFlow, which would decrease training times therefore it was chosen for this project.

#### **4.1.7. Visual Studio**

Visual Studio is Microsoft's IDE used to develop various programs and applications, it has support for almost all major programming languages. All the C# scripts in this project were written in Microsoft Visual Studio. Since Visual Studio can also be used for development on the .NET platform and is also compatible with the Unity Engine, there was no hesitation in choosing this IDE for the development of this project.

#### **4.1.8. Anaconda**

Anaconda is an open source distribution of Python, used mainly for scientific computing, data processing on a huge scale and predictive analytics. This is chosen for ease of access, since Anaconda is mainly used for scientific computing, the python distribution comes with all the necessary packages that are being used in the project, so there is no need to manually install all the required packages.

#### **4.1.9. Python**

The Python programming language was mainly used because TensorFlow is a Python library. Machine Learning is mostly implemented using Python because it is a very flexible language which allows for many options when it comes to development. In this project the agent is trained by using a python script.

#### **4.1.10 C#**

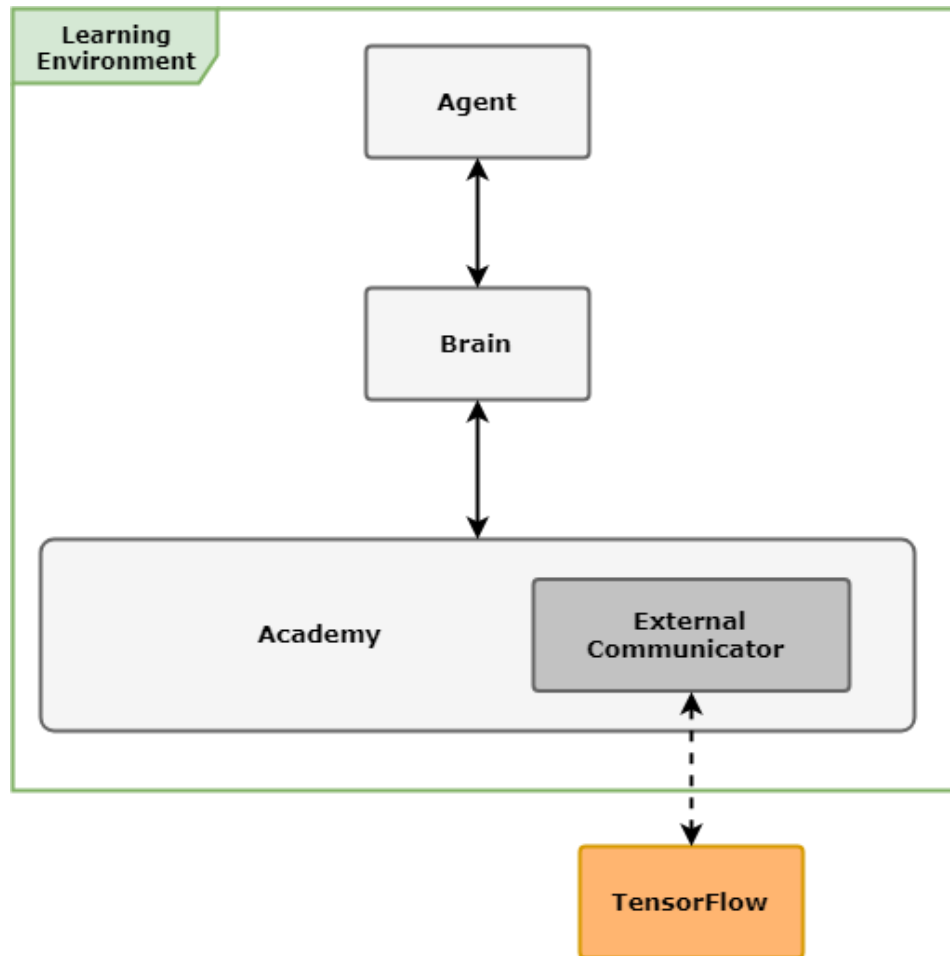
Since this project is based around the Unity engine and the Unity ML Agents, using C# was the optimal choice. Unity mainly uses C#; therefore, it was decided to be used from a very early stage. All of the scripting for the scene and the agent is done using C#.

### **4.2. Machine Learning Environment**

#### **4.2.1. Environment Structure**

The first thing needed to properly understand the implementation is to know how the learning environment is structured and what roles do the objects inside that environment play. There are three main objects inside this learning environment, the agent, the brain and the academy, this can be seen in the figure below. The figure also shows a visual illustration of how the learning environment is structured in this project.





**Figure 7 – Visual illustration of the learning environment.**

All the objects have their own scripts and are interconnected to each other, as it can be seen in the figure. The agent is attached to the main agent body that learns, it has unique observations and states and it receives rewards based on what action it takes. The brain is linked to the agent and is responsible of what action the agent takes. The academy is a parent of the brain and is responsible for defining the scope of the learning environment.

Understanding the roles of these three objects is crucial to properly understand, not only the implementation of the whole system but also these three objects on their own.

#### **4.2.2. Agent**

The agent implementation consists of three steps. First the agent collects observations from the environment, the agent is then given positive or negative rewards based on the action it takes

and the results of that action. Finally, after the agent completes its action and has received its rewards, the agent and the game state are reset to the initial starting state.

The agent is told what observations to collect from the environment using the `CollectObservations()` method. Figure 8 below, shows a code snippet of this method.

```
public override void CollectObservations()
{
    AddVectorObs(gameObject.transform.position.x);
    AddVectorObs(body.transform.position.x);
    AddVectorObs(platform.transform.position.x);
    AddVectorObs((body.transform.position.x - platform.transform.position.x));
    AddVectorObs((gameObject.transform.position.x - platform.transform.position.x));
    AddVectorObs((body.transform.position.x - gameObject.transform.position.x));
}
```

**Figure 8 - The `CollectObservation()` method.**

In this case the agent collects six different observations from the environment these are as follows, respectively:

- The position of the agent body on the X axis.
- The position of the enemy body on the X axis.
- The position of the platform on the X axis.
- The position difference between the enemy body and the platform on the X axis.
- The position difference between the agent body and the platform on the X axis.
- The position difference between the enemy body and the agent body on the X axis.

These observations act as sensors for the agent therefore most of the actions taken by the agent in the environment are based on them.

The actions for the agent are then set using the AgentAction() method. Figure 9 below, shows a code snippet of this method.

```
public override void AgentAction(float[] vectorAction, string textAction)
{
    if (brain.brainParameters.vectorActionSpaceType == SpaceType.continuous)
    {
        float action_right = 2f * (vectorAction[0]);
        float action_left = 2f * (vectorAction[1]);

        if (action_right > 0f)
        {
            RigBod.velocity = new Vector3(move, RigBod.velocity.y);
        }
        else if (action_left < 0f)
        {
            RigBod.velocity = new Vector3(-move, RigBod.velocity.y);
        }
        else RigBod.velocity = new Vector3(0, RigBod.velocity.y);
    }

    if (
        (body.transform.position.y - platform.transform.position.y) < -20f ||
        Mathf.Abs(body.transform.position.x - platform.transform.position.x) > 25f ||
        Mathf.Abs(body.transform.position.z - platform.transform.position.z) > 25f
    )
    {
        Done();
        SetReward(1f);
    }

    if (
        (gameObject.transform.position.y - platform.transform.position.y) < -40f ||
        Mathf.Abs(gameObject.transform.position.x - platform.transform.position.x) > 40f ||
        Mathf.Abs(gameObject.transform.position.z - platform.transform.position.z) > 40f
    )
    {
        Done();
        SetReward(-1f);
    }
}
```

Figure 9 - The AgentAction() method.

In this implementation the agent is given two action states:

- float action\_right = 2f \* (vectorAction[0]);
- float action\_left = 2f \* (vectorAction[1]);

These action states move the agent right or left, respectively. The if statements decide whether the value set to the states is greater than 0 (Right) or less than 0 (Left), if any of those conditions meet, the agent body performs the corresponding action. The rewards are also set within this method using SetReward(); , in the case of this implementation, if the enemy body falls out of the range of the platform the agent is to receive a **positive reward** and if the agent

body falls out of the range of the platform the agent is to receive a **negative reward**. After one of the conditions is met the session ends and the agent is given the corresponding reward.

Finally, when the session ends the agent is reset back to its initial state using the AgentReset() method. Figure 10 below, shows a code snippet of this method.

```
public override void AgentReset()
{
    gameObject.transform.position = new Vector3(0f, 0f, 0f);
    bodyStartPos = UnityEngine.Random.Range(-20f, 20f);
    body.transform.position = new Vector3(bodyStartPos, 0, 0);
    body.GetComponent<Rigidbody>().velocity = new Vector3(0f, 0f, 0f);
}
```

**Figure 10 – The AgentReset() method.**

This method is straight forward, there are three main things going on in this method. First, the agent body's position is reset to its initial starting position. Second, the enemy body's position is randomized and set so that on every reset, it is in a different position from the agent, either to the right of the agent body or to the left. Lastly, the movement of the enemy body is set to 0, otherwise the enemy body would keep moving even after the session is reset.

#### **4.2.3. Brain**

As mentioned before the brain is linked to the agent, its role is to decide the agent's actions. The brain can make that decision through four different mediums, it has a separate mode for each of these mediums, that can be changed in the Unity engine according to the developer's needs. These modes are Player, Heuristic, External and Internal<sup>[20]</sup>.

- **Player:** In this mode the decisions for the actions of the agent are made by the player using different inputs.
- **Heuristic:** The action decisions in this mode are made based on behavior that is coded by the developer.

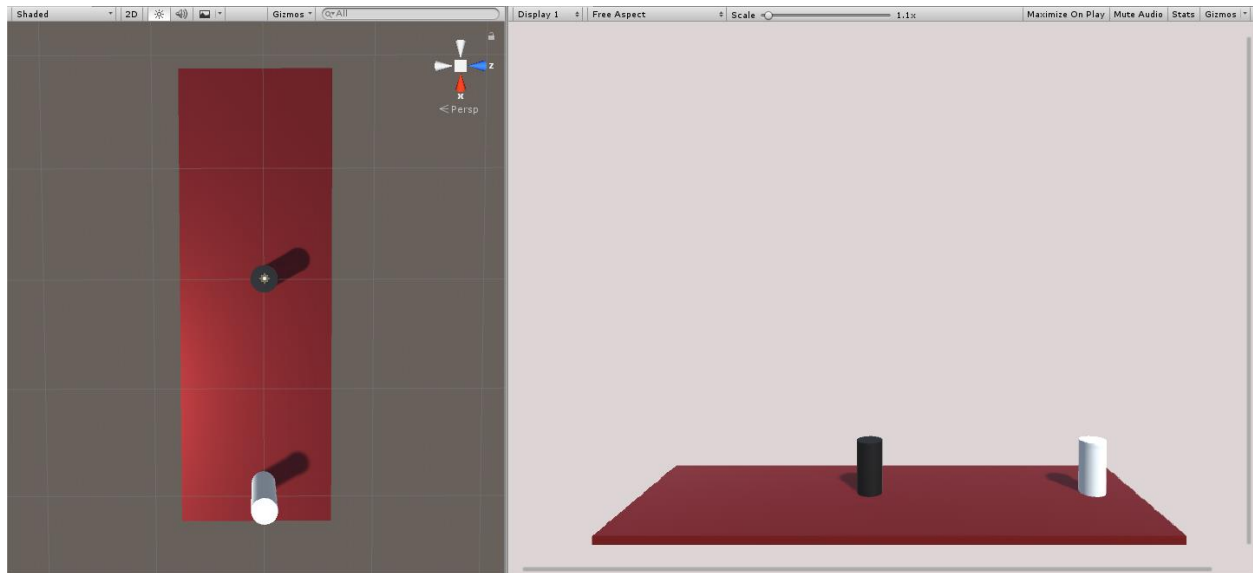
- **External:** The action decisions in this mode are made through an external communicator using a Machine Learning library, in this case TensorFlow.
- **Internal:** In this mode the actions of the agent are decided using a trained model. In the case of this implementation the model is generated after the training process, by TensorFlow, which is then embedded into the brain through TensorFlowSharp.

#### **4.2.4. Academy**

The academy contains the brain as a child in an environment, it orchestrates the agent's observations and all the decision-making processes. The academy can be used to specify various environmental parameters, for example, the environment's run speed. In the case of this implementation the academy was left at default, for a fair evaluation of the agent. When it is run, it can be in two modes, training mode or inference mode. The mode is selected based on the presence of an external communicator, so if the brain mode is set to external the academy is in training mode and if the brain mode is set to internal the academy is in inference mode<sup>[30]</sup>.

### **4.3. Game**

The game is entirely developed in the Unity engine and is fairly easy to understand, there is one scene, made up of a platform and 2 cylinder objects that are the agent body (Black) and the enemy body (White). The goal of the game is to move the agent body left or right to knock the enemy body off the platform. Each time the game is ran, the scene is set with the agent body in the middle of the platform and the enemy body at a random location on the platform, either on the right side or the left side of the agent body. The image below shows a snapshot of the game.



**Figure 11 – Snapshot from the game**

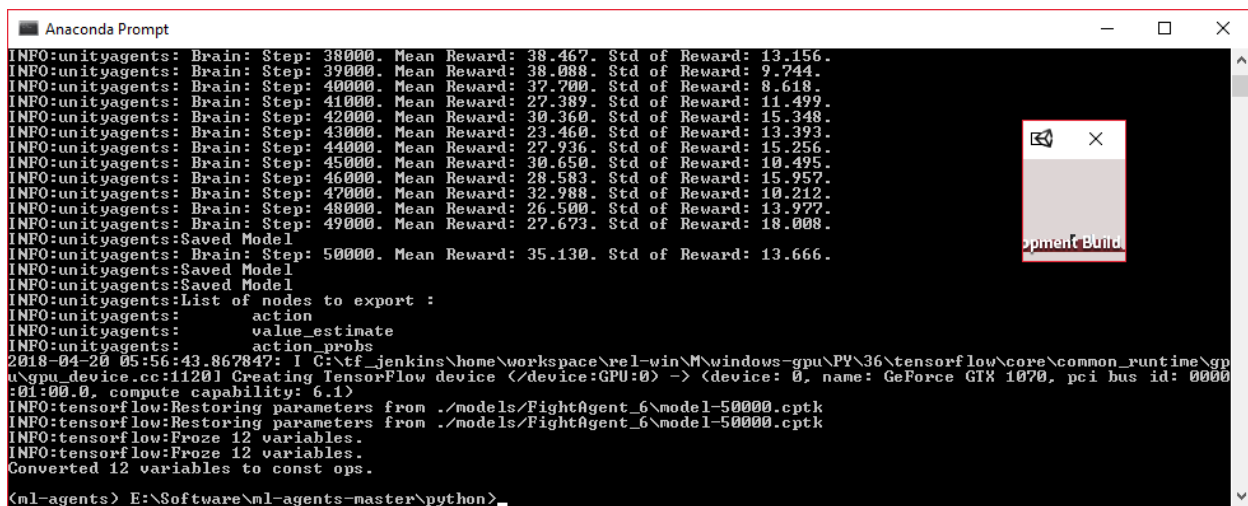
Initially the game was going to include more functionality, such as kicking and punching but due to difficulties with the SDKs and Libraries, the game was made simpler, to focus more on the machine learning implementation rather than making the game itself.

#### **4.4. Training the agent**

The agent is trained using Proximal Policy Optimization (PPO), a reinforcement learning algorithm that the Unity ML-Agents SDK uses. This algorithm is shown to be more efficient compared to many other Reinforcement Learning algorithms. Before the training process can start the environment needs to be set up from the Unity engine. Firstly, the brain is set to external mode and the game is built so that an executable file can be obtained. The executable file must then be placed within the python directory inside the ML-Agents SDK, the location of this executable file is crucial because the training process uses different scripts from within that same directory. Finally, to begin the training process, Anaconda prompt is launched, and the directory is changed into the same directory the executable file was placed in. After that is done the following command is inputted

```
python learn.py <Path of executable file> --run-id=< name for the model> --train
```

This command runs a python wrapper script that contains the constants and parameters needed for the training environment to train the agent. The agent then runs and plays the game for a certain amount of times before it saves a graph model of the trained agent, each run is called a step, at default this is done for 50000 steps. The image below shows the training process after it is completed.



```
Anaconda Prompt
INFO:unityagents: Brain: Step: 38000. Mean Reward: 38.467. Std of Reward: 13.156.
INFO:unityagents: Brain: Step: 39000. Mean Reward: 38.088. Std of Reward: 9.744.
INFO:unityagents: Brain: Step: 40000. Mean Reward: 37.700. Std of Reward: 8.618.
INFO:unityagents: Brain: Step: 41000. Mean Reward: 27.389. Std of Reward: 11.499.
INFO:unityagents: Brain: Step: 42000. Mean Reward: 30.360. Std of Reward: 15.348.
INFO:unityagents: Brain: Step: 43000. Mean Reward: 23.460. Std of Reward: 13.373.
INFO:unityagents: Brain: Step: 44000. Mean Reward: 27.936. Std of Reward: 15.256.
INFO:unityagents: Brain: Step: 45000. Mean Reward: 30.650. Std of Reward: 10.495.
INFO:unityagents: Brain: Step: 46000. Mean Reward: 28.583. Std of Reward: 15.957.
INFO:unityagents: Brain: Step: 47000. Mean Reward: 32.988. Std of Reward: 10.212.
INFO:unityagents: Brain: Step: 48000. Mean Reward: 26.500. Std of Reward: 13.977.
INFO:unityagents: Brain: Step: 49000. Mean Reward: 27.673. Std of Reward: 18.008.
INFO:unityagents: Saved Model
INFO:unityagents: Brain: Step: 50000. Mean Reward: 35.130. Std of Reward: 13.666.
INFO:unityagents: Saved Model
INFO:unityagents: Saved Model
INFO:unityagents: List of nodes to export :
INFO:unityagents:      action
INFO:unityagents:      value_estimate
INFO:unityagents:      action_probs
2018-04-20 05:56:43.867847: I C:\tf_jenkins\home\workspace\rel-win\M\windows-gpu\PY\36\tensorflow\core\common_runtime\gp
u\gpu_device.cc:1120] Creating TensorFlow device (/device:GPU:0) -> /device: 0, name: GeForce GTX 1070, pci bus id: 0000
:01:00.0, compute capability: 6.1)
INFO:tensorflow:Restoring parameters from ./models/FightAgent_6\model-50000.cptk
INFO:tensorflow:Froze 12 variables.
INFO:tensorflow:Froze 12 variables.
Converted 12 variables to const ops.
(ml-agents) E:\Software\ml-agents-master\python>
```

**Figure 12 – Training process**

As it can be seen in the image, the prompt displays the rewards the agent gets. At each step the agent's mean reward is calculated, based on that the agent tries to improve and tries to get a higher mean reward. More reward means better learning which in turn means that the agent is improving.

As mentioned before, at the end of each training session a graph model of the trained agent is saved. By switching the brain to internal mode, this graph model can now be embedded into the brain and be freely used by the agent in the game, without the need of TensorFlow (or an external communicator).

## 5. Evaluation

### 5.1. Evaluation Methodology

The success of the implementation is evaluated by comparing three cases, where each case has a different reward system for the agent. All three of these cases are put through 10 training sessions. The training sessions determine the overall effectiveness and learning speed of the agent while the cases determine the most effective reward system for the agent. A comparison is made on the basis of the agent's summary statistics that are collected during the training process. These statistics are Entropy and Cumulative reward <sup>[32]</sup>.

- **Entropy**: The randomness of the agent's decisions while its training. Entropy should generally decrease on successful training sessions.
- **Cumulative reward**: The cumulative mean reward that the agent has received. As the agent gets better the mean reward generally increases.

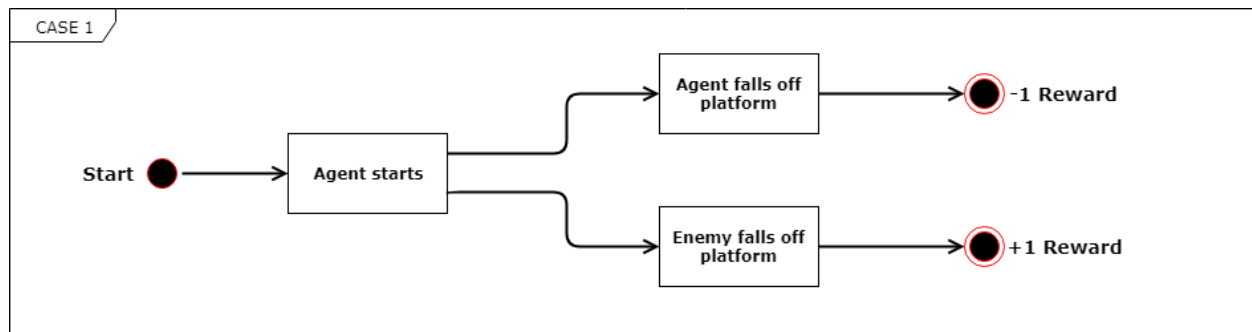
These statistics are visualized via a line graph and collected using TensorBoard. To further reinforce the accuracy of results gathered from the summary data, the agent is tested by embedding the trained model of each case into the brain and counting the number of successes during 60 seconds of runtime.

### 5.2. Evaluation cases

#### 5.2.1. Case 1

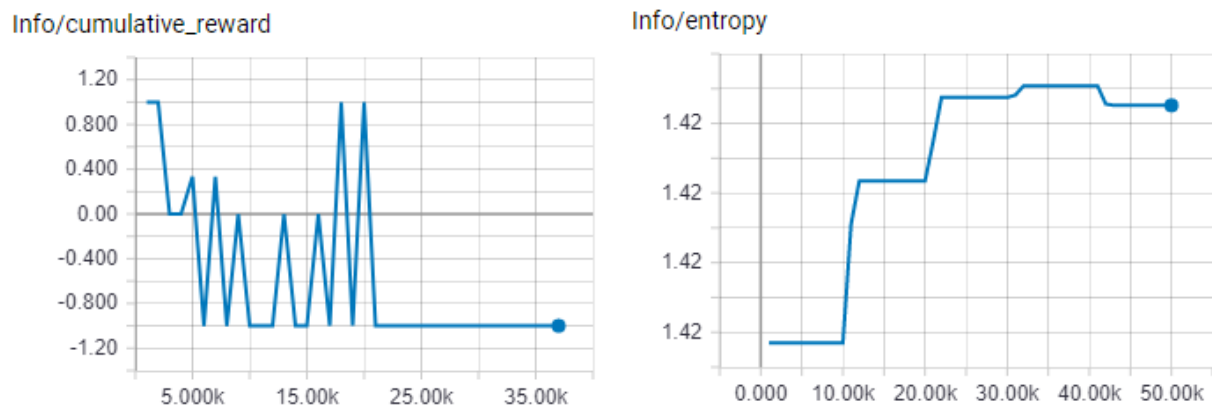
In this case the agent is only given two kinds of rewards. A positive reward of 1 for pushing the enemy body off the platform and a negative reward of 1 for falling off the platform. The diagram below illustrates how the reward system works for this case.





**Figure 13 – Reward system for case 1**

### Training session 1



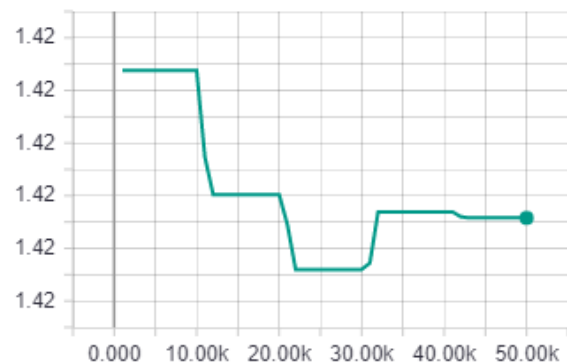
As shown by the graphs above, the first training session of case 1 did not produce any significant results. The agent's cumulative reward (mean reward) fluctuates a lot and then stabilizes in the negative region, this means that the agent is getting more negative rewards which should not be the case. The entropy increases, in other words the randomness of the agent's decision keeps increasing, this means the agent is making random decisions without any significant results.

## Training session 5

Info/cumulative\_reward



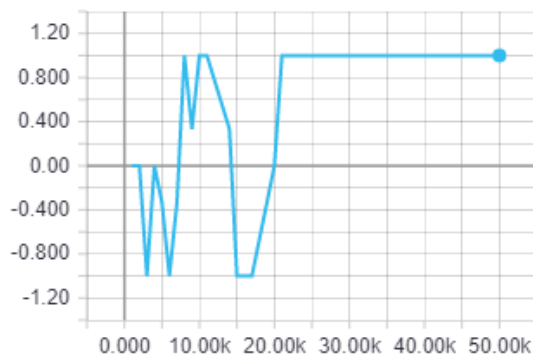
Info/entropy



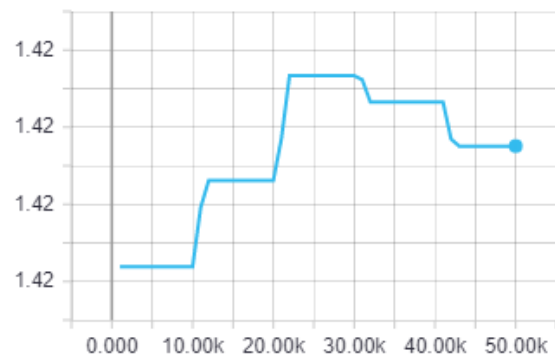
The fifth training session is already much better than the first one. As shown by the graphs above, the agent's cumulative reward still fluctuates but unlike the first session, it ends in the positive region which shows that the agent is at least getting some positive reward. The entropy decreases which is much better compared to the first session, this shows that the agent is starting to understand what gives it positive rewards and what gives it negative rewards.

## Training session 10

Info/cumulative\_reward



Info/entropy



As it can be seen by the graphs, by the tenth session the agent already has a stable positive cumulative reward throughout 30,000 steps which means that the agent had a consistent mean reward. The entropy increased to its peak and then starts decreasing which is a good thing.

Over the span of these three training sessions we can see that using the reward system of case 1 the agent requires multiple trainings before it reaches a stable state. The learning starts of slow but after several training sessions it becomes stable.

### 5.2.2. Case 2

In case 2 the agent is given the same rewards as case 1, but with the addition of one more reward. The agent is given an additional positive reward of 0.1 if any movement is made, this can be seen in the diagram below.

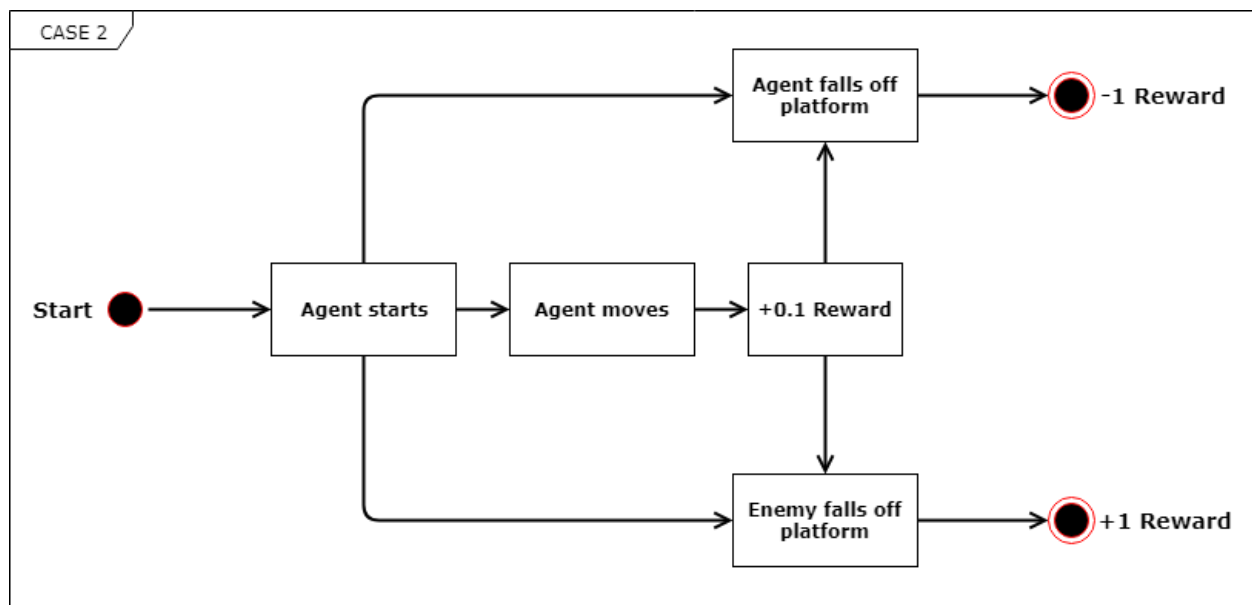
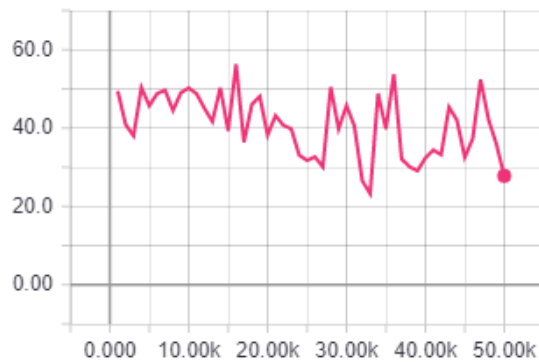


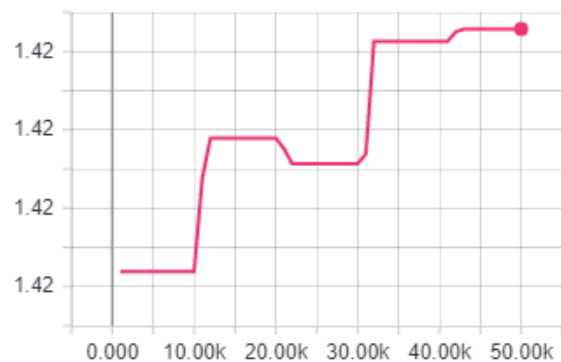
Figure 14 – Reward system for case 2

## Training session 1

Info/cumulative\_reward



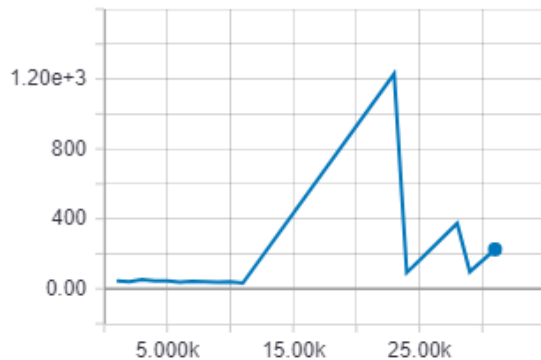
Info/entropy



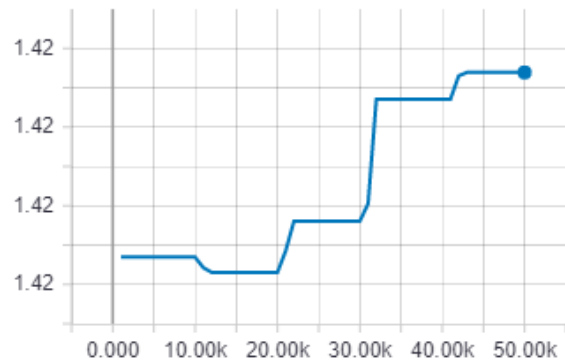
Based on the graphs above the agent's cumulative reward is positive and stays approximately between the range of 30 and 50, which is a good only if the entropy decreases. This is because the entropy is the randomness of the agent's decisions and in this training session the agent is increasingly making random decisions which means that most of the rewards the agent is getting is either based on luck or because of the positive reward given due the agent's movement.

## Training session 5

Info/cumulative\_reward

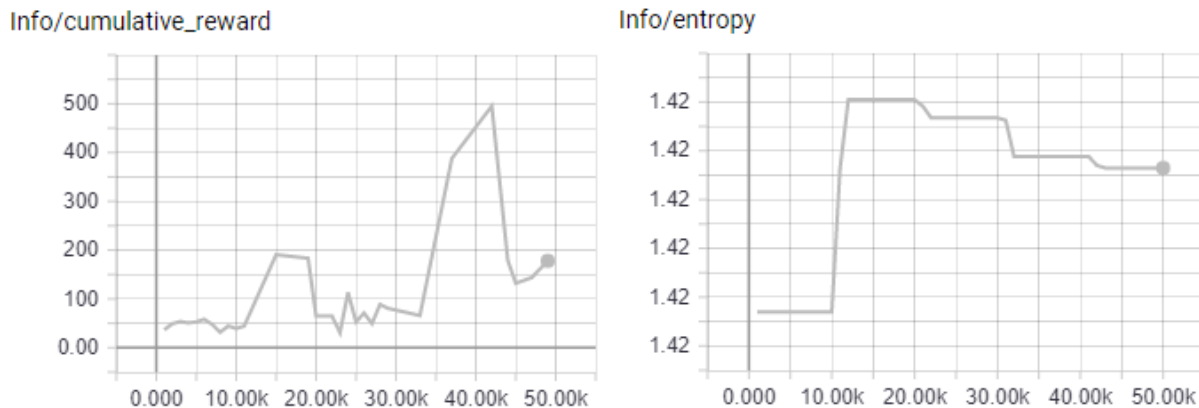


Info/entropy



The results for this session are very strange, the agent's cumulative reward increases at a steady pace but suddenly drops and starts fluctuating while the entropy steadily increases. This proves that the agent is receiving most of its rewards based on luck.

## Training session 10



This session looks very similar to the fifth training session, the cumulative reward is similar to the one in the fifth training session but in reverse, in this session it fluctuates first then increases at a steady pace and then suddenly drops. The entropy is at its peak but slightly decreases, which is better than the first and fifth training session in this case.

By comparing all three training sessions it can clearly be seen that using the reward system of case 2 results in an agent that just moves. Most of the reward the agent receives is because of the reward the agent gets when it moves, so the agent mostly learns to move rather than the main objective of the game, to knock the enemy off the platform.

### 5.2.3. Case 3

Case 3 further adds onto case 2 by adding another reward. The agent now receives a positive reward of 0.5 if the agent body touches the enemy body. The diagram below shows how the rewards are processed in case 3.

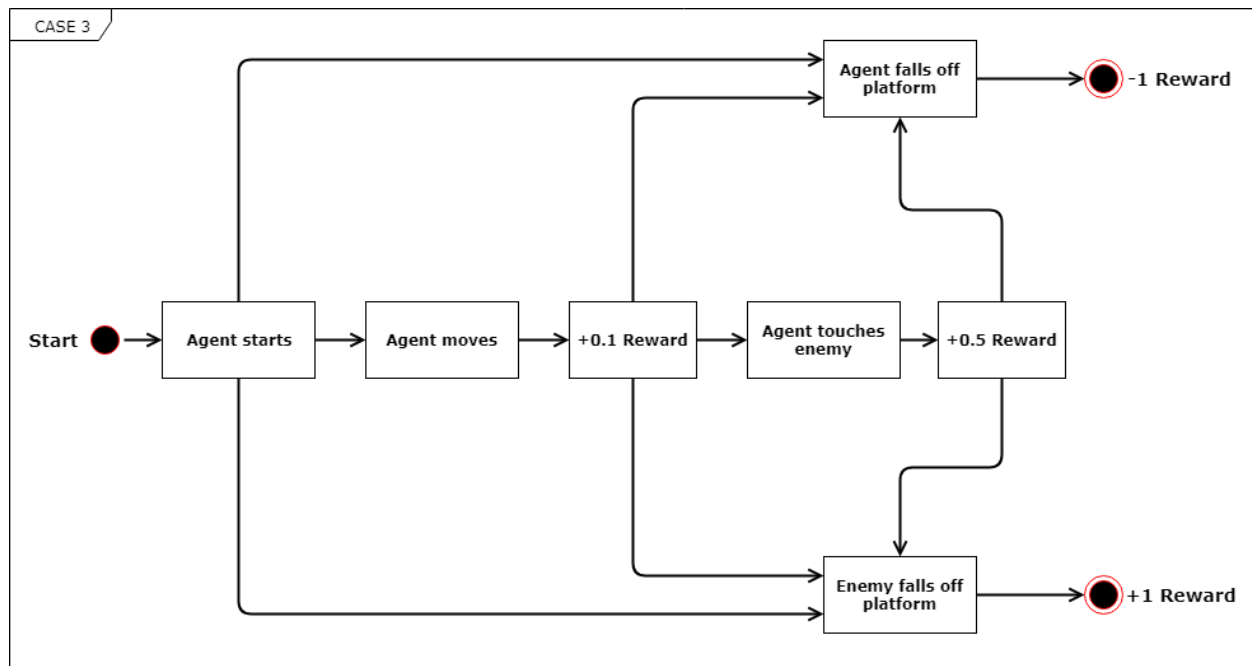
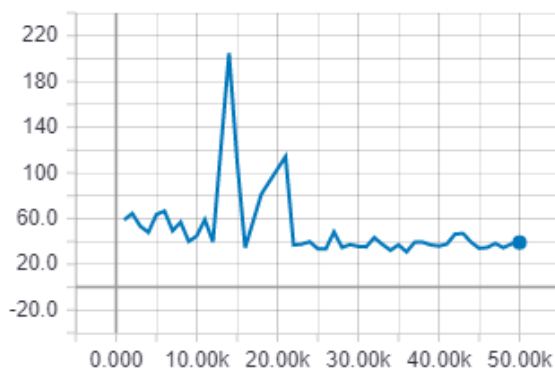


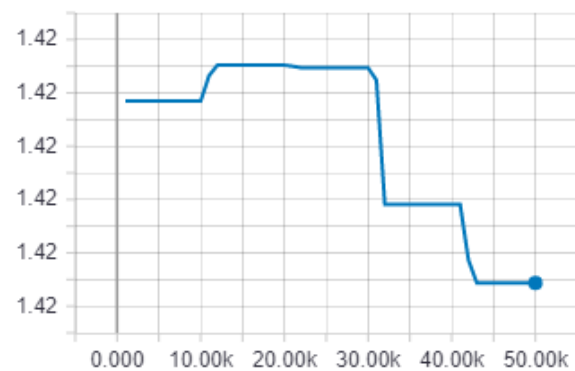
Figure 15 – Reward system for case 3

## Training session 1

Info/cumulative\_reward



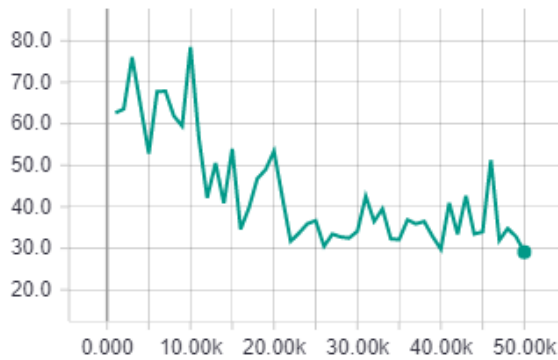
Info/entropy



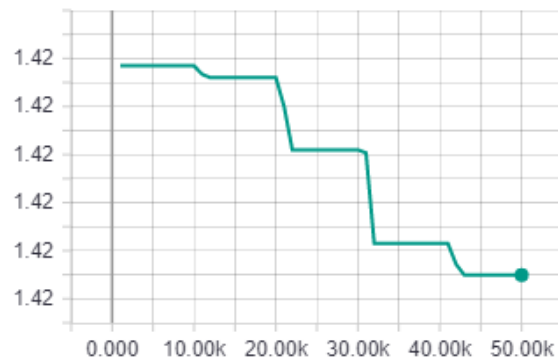
As seen by the graphs above, the agent's cumulative reward fluctuates in the beginning but gets almost completely stable, this means that the agent gets almost a consistent mean reward. The entropy decreases which means the randomness in the agent's decision decreases this results in an agent that knows what actions give it positive and negative rewards.

## Training session 5

Info/cumulative\_reward



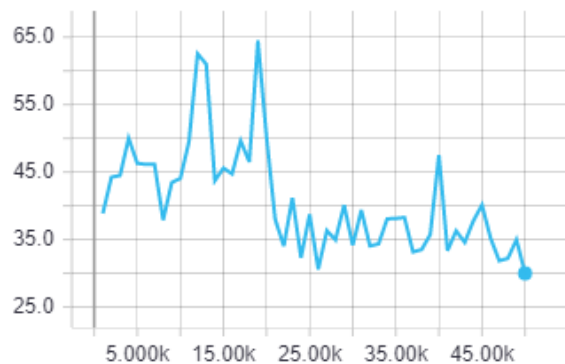
Info/entropy



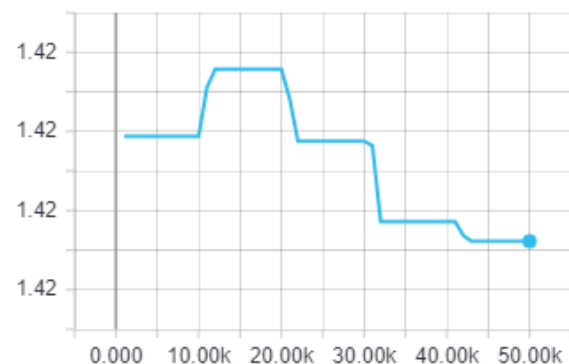
The cumulative reward fluctuates in this session, but it ends around the same place the first session ended at. The entropy also decreases in this session. The results for this session are very similar to the first training session.

## Training session 10

Info/cumulative\_reward



Info/entropy



As shown by the graphs above, the cumulative reward fluctuates even more than the fifth training session, but it ends up at around the same place the first and fifth sessions ended at. Just like the first and fifth session the entropy in this session also decreases.

By comparing all three of the training sessions in case 3, the reward system used in this case seems to be the optimal one. The agent did not take many sessions to learn and received

almost a consistent mean reward throughout all the training sessions making this an effective reward system.

## **5.3. Outcome**

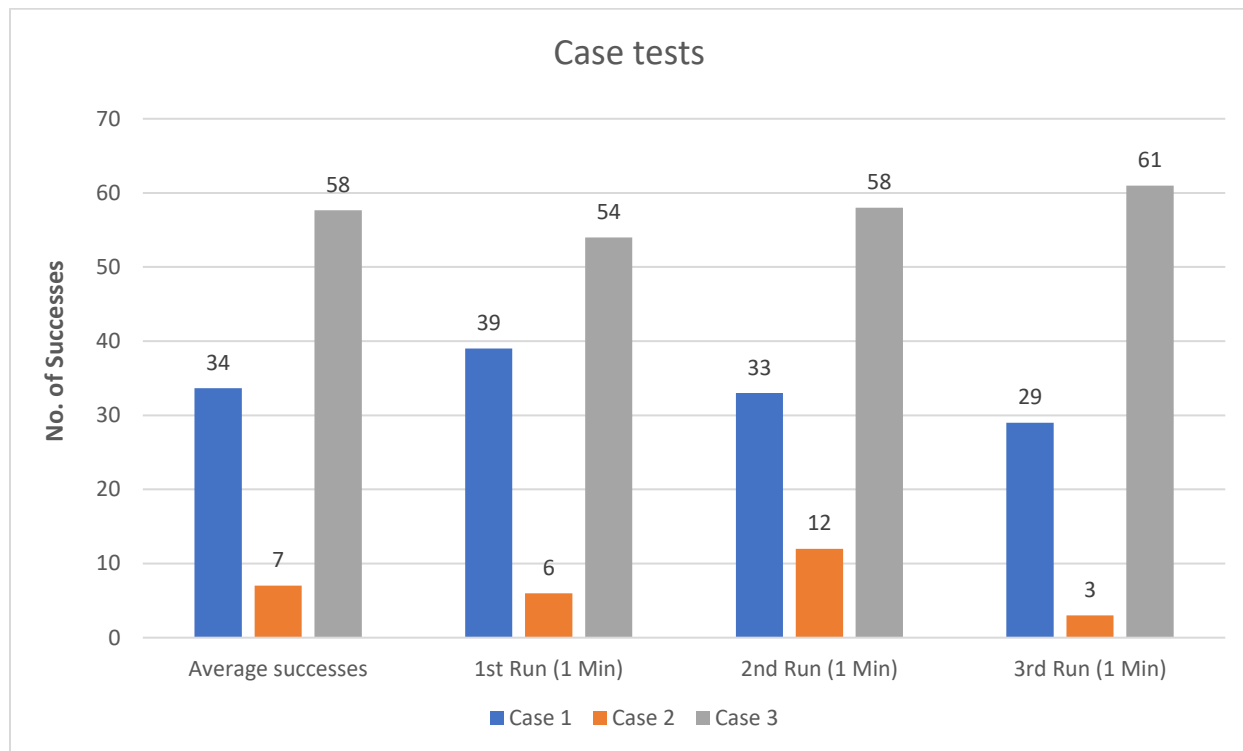
### **5.3.1. Case Comparison**

All three of the cases produced different results but the outcome in all three of these cases was the same. By the tenth training session in cases 1 and 3, it can clearly be seen that the agent decreases the amount of random decisions it makes and eventually learns to perform the actions it is given. The same is not true for case 2 in which the agent keeps making random decisions and gets left with a positive mean reward because of the reward system used. This does not mean that case 2 will not work, this just means that case 2 is very inefficient.

The reward system for Case 1 produced efficient results but it was slow for the agent to learn. It took 10 training sessions to achieve a positive cumulative reward. Whereas the reward system for Case 2 was fast to **train** but very inefficient at achieving the objective of the game. Finally, the reward system used in Case 3 seems to be the best one, it is both fast for the agent to learn and very efficient at completing the objective of the game.



### 5.3.2. Testing the cases



To further reinforce the credibility of the training process and reward systems, the agent's brain was embedded with the trained graph model obtained from the 10<sup>th</sup> training session of each case. Each case model was ran for exactly 1 minute while noting down the number of times the agent successfully pushed the enemy body off the platform. This process was repeated 3 times and then using the data collected, an average number of successes was calculated.

The bar chart above shows that Case 3 has the most number of successes in 1 minute whereas Case 2 has the least amount of successes. Case 1 seems to be in the middle but as mentioned before it is not fast.

### 5.3.3. Justifying test results

The agent does not learn fast using case 1 because of the simplicity of its rewards system. The agent gets +1 for knocking the enemy off the platform and -1 for falling off the platform. In the beginning the agent falls off a few times and realizes that falling off is bad therefore avoids it,

but since there is no positive reward other than knocking off the enemy, the agent must do random actions until it eventually knocks the enemy off the platform and gets a positive reward. Once the agent gets a positive reward it realizes that it is a good thing and thus tries to do it again to get more positive rewards. The agent keeps going through this slow trial and error process which results in slower learning times.

Case 2 has the least amount of success numbers because the agent is given a reward of +0.1 just for moving. In the early stages of learning the agent gets negative rewards for falling off the platform, it realizes that falling off is bad and therefore avoids it. Since the agent keeps getting positive reward for movement it realizes that moving is a good thing and therefore keeps randomly moving on the platform, ignoring the main objective of the game.

Case 3 works best because of the reward system it uses. The agent is given +1 for knocking the enemy off the platform, -1 for falling off the platform, +0.1 for moving and +0.5 for touching the enemy. This is the same as case 2 but the only difference is that there is an additional positive reward that is greater than the movement reward, this gives the agent an extra option which helps it not to get trapped in one reward. In the beginning the agent falls off and gets negative rewards, realizing that it is bad to fall off the agent moves randomly to gain positive rewards, as soon as the agent comes close to the enemy it realizes that touching the enemy is a much higher reward than moving so it actively starts searching for the enemy. Finally, at some point the agent knocks the enemy off the platform and realizes that is an even higher reward than touching the enemy and therefore continues to do that. This concludes that the reward system used in case 3 produces a well learned agent that completes the game's objective of knocking the enemy off the platform, learns faster and is much more effective compared to the other two cases.

## **6. Conclusion**

### **6.1. Project Summary**

To conclude, the project was a success because machine learning was successfully implemented in a 2D fighting game. This was accomplished using a recently released SDK by Unity Engine called ML-Agents, that is, at the time of writing this documentation, still in the beta testing phase. The learning agent is successfully able to play the developed game autonomously with a high degree of precision.

Even though setting up the environment and the libraries took more time than anticipated, the implementation turned out to be very interesting and allowed me to dive further deep into Machine Learning technologies.

### **6.2. Project Management**

The project was carried out with regards to the project plan set up in the first deliverable and an agile methodology. The progress of the project has been mostly matched with the plan outlined in the first deliverable. Progress slowed down when reaching the end due to some issues with properly setting up libraries and environments, but it was effectively managed to stay on track.

### **6.3. Project Outcomes**

The aims of this study, from section 1.1 have mostly been met. An AI agent has been developed using Machine Learning Techniques that is able to learn a 2D fighting game specifically developed for the agent. As per the aims of this study, a video game has also been developed specifically for the learning agent. Although, not all functionalities mentioned could be implemented into the video game, the main aim for this study was to implement Machine Learning in a video game, and this has been achieved.

As for the objectives of this project from section 1.2, all of the objectives have been completed. An AI agent that can learn the core elements of the game has been successfully developed along with a 2D fighting game for the agent. One of the objectives requires the agent to be flexible and be able to adapt to any changes made to the game. Since the agent works based on observations collected from its environment, the agent is highly adaptable. Any changes made to the game can be stated in the agent's observations and the agent would adapt to it. Finally, the last objective requires that the agent be capable of running autonomously until manually stopped. After the training process the agent goes through using TensorFlow a graph model of that trained agent is generated, that can be imbedded into the brain in the Unity Engine which allows the agent to run autonomously until manually stopped, this has been described in section 4.4.

Along with that almost all of the important functional requirements (refer to Appendix A1) from the first deliverable have also been met, the only requirements that have not been met are the ones related to game functionality and as mentioned before this study mainly focuses on the Machine Learning implementation.

## **6.4. Professional, Legal and Ethical Issues**

### **6.4.1. Professional Issues**

It is my responsibility to ensure that that this project is implemented in a professional manner in accordance to good practice. As the author of this project, I accept full responsibility for the work within this project and any consequences that result with it. As a student representing Heriot Watt University, It will be made sure that no harm comes to the university's reputation as a result of unprofessional stands. In regard to this, the Heriot Watt University Code of Good Practice in Research, should be brought to attention which will be heavily taken into acknowledgement during this project.

### **6.4.2. Legal Issues**

As the project involves the use of external software currently under license, all of its use and conduct will be taken into consideration and strictly followed.

### **6.4.3. Ethical Issues**

Information Technology (IT) professionals are often accused of being obsessed with functionality, mostly at the cost of ethical issues. In this project an AI learning agent will be created for a simple video game, ethical issues here will be taken into consideration as the learning agent might be seen as bullying players. There will be special care taken during the development of the learning agent for this to not happen. The Heriot Watt University Code of Good Practice in Research specifically talks about the data protection act and ethical principles in research projects that involve human participants. As testing with human participants is planned there will be care taken to follow the principles and make sure all data collected is anonymous.

## **6.5. Reflection and Future Work**

Overall, the machine learning implementation was a success. The agent functions as it is intended to with no issues. Looking back at the risk assessment (refer to Appendix A2) from deliverable one there were many mitigation circumstances that could have taken place throughout the development process for both the game and the agent, but the only thing that put this project under risk was the libraries being unstable. The latest TensorFlow libraries were not supported by the Nvidia CUDA Toolkit 8, finding a fix took a lot of time and with deadlines approaching fast it was decided that some of the functionalities of the game would be stripped away so that there could be more time to focus on the Machine Learning implementation. The functionalities that were stripped away from the game were kick, punch and block, the game was instead made a little simpler while fully maintaining the functionality of the agent and

keeping the fighting game genre, in this game the agent would try and push the enemy off the platform, like sumo wrestling.

There are many things that could be expanded in the future. With more time a game with all the functionalities mentioned could be developed. One of the major requirements of machine learning is time. Training a complex agent takes up a lot of time. This project uses just one of the many types of learning techniques included within the ML-Agents SDK, it includes many more interesting learning techniques that can be used to create complex things like an evolutionary learning agent. With more time given ML-Agents could be explored in more depth and also there would be the possibility of developing more complex agents that could learn by copying the player using imitation learning.

## 7. References

- [1] Ertel, W. (2011). Introduction to Artificial Intelligence. Wiesbaden: Vieweg+Teubner.
- [2] Copeland, J. (2000). What is Artificial Intelligence?. [online] AlanTuring.net. Available at: [http://www.alanturing.net/turing\\_archive/pages/reference%20articles/what%20is%20ai.html](http://www.alanturing.net/turing_archive/pages/reference%20articles/what%20is%20ai.html) [Accessed 31 Oct. 2017]
- [3] Executive Office of the President National Science and Technology Council Committee on Technology, (2016). PREPARING FOR THE FUTURE OF ARTIFICIAL INTELLIGENCE. [online] Available at: [https://obamawhitehouse.archives.gov/sites/default/files/whitehouse\\_files/microsites/ostp/NSTC/preparing\\_for\\_the\\_future\\_of\\_ai.pdf](https://obamawhitehouse.archives.gov/sites/default/files/whitehouse_files/microsites/ostp/NSTC/preparing_for_the_future_of_ai.pdf) [Accessed 31 Oct. 2017]
- [4] Hammond, K. (2015). Practical Artificial Intelligence For Dummies. Hoboken: John Wiley & Sons, Inc, Pg 5-10.
- [5] IBM100. (2017). Deep Blue. [online] Available at: <http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/> [Accessed at 31 Oct. 2017]
- [6] M. Wooldridge and N. R. Jennings. (1995) Intelligent agents: Theory and practice. The Knowledge Engineering Review, 10(2):115–152.
- [7] Russell, Stuart J.; Norvig, Peter (2003), Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, ISBN 0-13-790395-2, chpt. 2
- [8] Lee, John Ray. (2006). Conversations with an intelligent agent: modeling and integrating patterns in communications among humans and agents. PhD (Doctor of Philosophy) thesis, University of Iowa.
- [9] Chadha, N., Payce, J. and Roy, T. (2006). Agents Artificial Intelligence. [online] Available at: <https://www.doc.ic.ac.uk/project/examples/2005/163/g0516334/index.html> [Accessed at 2 Nov. 2017]
- [10] Learneroo, (2016). Reflex Agents. [online] Available at: <https://www.learneroo.com/modules/89/nodes/469> [Accessed 2 Nov. 2017]

- [11] Mills, F. and Stufflebeam, R. (2005). Introduction to Intelligent Agents. [online] Consortium on Cognitive Science Instruction. Available at:  
[http://www.mind.ilstu.edu/curriculum/ants\\_nasa/intelligent\\_agents.php](http://www.mind.ilstu.edu/curriculum/ants_nasa/intelligent_agents.php) [Accessed 2 Nov. 2017]
- [12] Rasmussen, J. (2016). Are Behavior Trees a Thing of the Past?. [Blog] Gamasutra. Available at:  
[https://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are\\_Behavior\\_Trees\\_a\\_Thing\\_of\\_the\\_Past.php](https://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are_Behavior_Trees_a_Thing_of_the_Past.php) [Accessed 4 Nov. 2017]
- [13] Hoorn, N., Togelius, J., Wierstra, D. and Schmidhuber, J. (2009). Robust player imitation using multiobjective evolution. Manno: Dalle Molle Institute for Artificial Intelligence (IDSIA). Available at: <http://people.idsia.ch/~daan/papers/cec2009.pdf> [Accessed 4 Nov. 2017]
- [14] Samuel, A. (1959) Some Studies in Machine Learning Using the Game of Checkers. IBM Journal, [online] July 1959. Available at:  
<http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5392560> [Accessed 4 Nov. 2017]
- [15] Kohavi, R. and Provost, F. (1998). Special Issue on Applications of Machine Learning and the Knowledge Discovery Process. Machine Learning [online] 30, 271-274. Available at:  
<http://robotics.stanford.edu/~ronnyk/glossary.html> [Accessed 6 Nov. 2017]
- [16] Alpaydin, E. (2009). Introduction to Machine Learning. Second Edition. Cambridge: The MIT Press.
- [17] Copeland, M. (2016). What's the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning?. [online] Nvidia. Available at:  
<https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/> [Accessed 6 Nov. 2017]
- [18] Mohri, M., Rostamizadeh, A. and Talwalkar, A. (2012). Foundations of Machine Learning. Cambridge: The MIT Press.



- [19] SethBling. (2015). Marl/O - Machine Learning for Video Games. [video] Available at: <https://www.youtube.com/watch?v=qv6UVOQ0F44> [Accessed 6 Nov. 2017]
- [20] Juliani, A. (2017). Introducing: Unity Machine Learning Agents. [Blog] Unity Blogs. Available at: [https://blogs.unity3d.com/2017/09/19/introducing-unity-machine-learning-agents/?\\_ga=2.129396367.2062437133.1509142812-320300202.1509142812](https://blogs.unity3d.com/2017/09/19/introducing-unity-machine-learning-agents/?_ga=2.129396367.2062437133.1509142812-320300202.1509142812) [Accessed 9 Nov. 2017]
- [21] Miikkulainen, R. (2006). Creating Intelligent Agents in Games. [pdf] Available at: <https://pdfs.semanticscholar.org/0f92/b7d293ee605ced9f7e6f28f5afe30073b149.pdf> [Accessed 9 Nov. 2017]
- [22] Chen, J. (2007). Flow in Games. Communications of the ACM magazine, Vol 50 (4), Pg31.
- [23] Enterprise Gamification. (2015) The concept of Flow introduced by Mihaly Csikszentmihalyi. [image] Available at: <http://www.enterprise-gamification.com/mediawiki/images/thumb/7/7a/FlowTheory.png/400px-FlowTheory.png> [Accessed 9 Nov. 2017]
- [24] Costikyan, G. (2013). Uncertainty in Games. Cambridge: The MIT Press, Pg9-Pg12.
- [25] Cheung, H. (2003). Computer Games. [pdf] Available at: <http://www3.ntu.edu.sg/home/asschui/Computer%20Games.PDF> [22 Nov. 2017]
- [26] Lueangrueangroj, S. and Kotrajaras, V. (2009). Real-Time Imitation Based Learning for Commercial Fighting Games. [online] Available at: [https://www.cp.eng.chula.ac.th/~vishnu/gameProg/papers/CGAT\\_Real%20Time%20Imitaion%20Based%20Learning.pdf](https://www.cp.eng.chula.ac.th/~vishnu/gameProg/papers/CGAT_Real%20Time%20Imitaion%20Based%20Learning.pdf) [Accessed 24 Nov. 2017].
- [27] tensorflow.org, (2018). TensorFlow Official Website. [online] Available at: <https://www.tensorflow.org/> [Accessed 18 Apr. 2018].

- [28] tensorflow.org, (2018). TensorBoard: Visualizing Learning. [online] Available at: [https://www.tensorflow.org/programmers\\_guide/summaries\\_and\\_tensorboard](https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard) [Accessed 18 Apr. 2018].
- [29] Ebersole, M. (2012). What is CUDA?. [online] NVIDIA. Available at: <https://blogs.nvidia.com/blog/2012/09/10/what-is-cuda-2/> [Accessed 19 Apr. 2018].
- [30] NVIDIA DEVELOPER. (2018). NVIDIA cuDNN. [online] Available at: <https://developer.nvidia.com/cudnn> [Accessed 19 Apr. 2018].
- [31] Github, (2018). Unity-Technologies/ml-agents. [online] Available at: <https://github.com/Unity-Technologies/ml-agents/blob/master/docs/ML-Agents-Overview.md> [Accessed 20 Apr. 2018].
- [32] Github, (2018). Unity-Technologies/ml-agents. [online] Available at: <https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Getting-Started-with-Balance-Ball.md> [Accessed 20 Apr. 2018].

## 8. Appendix

### A1 - Functional Requirements

ID	Requirements	
FR.1	The learning agent must be autonomous	Must Have
FR.2	The agent must be able to learn	Must Have
FR.3	The agent must be able to perform basic movement	Must Have
FR.4	The agent must be able to perform combat actions	Must Have
FR.5	The game must allow for basic movement	Must Have
FR.6	The game must allow for combact actions	Must Have
FR.7	The game should have a menu	Should Have
FR.8	The agent should be able to predict actions	Should Have
FR.9	The agent should be able to attack	Should Have
FR.10	The agent could be able to adapt its own play style	Could Have
FR.11	The game could have sound effects and music	Could Have

## A2 - Risk Analysis

Risk	Impact	Likelihood	Mitigation Strategy
Project is too complex or too big	Major	Likely	Reduce workload by decreasing functionality of project.
AI agent is unable to learn and perform all actions	Moderate	Likely	Decreaser functionality of project by focusing on one action.
AI agent is unable to learn and perform basic actions	Major	Unlikely	Decide on a simpler Machine learning technique.
Poor knowledge of developing may impact project plan.	Major	Likely	Reduce workload by focusing on the basic functionality.
Loss of Supervisor	Major	Rare	Arrange for another supervisor
AI agent may require high ammounts of computing power	Major	Unlikely	Decrease functionality of project to lower power usage.
Chosen Machine Learning technique does not intigrate well	Moderate	Likely	Find another Machine Learning technique.
Project plan negatively effected by semester 2 deadlines	Moderate	Likely	Reduce workload by decreasing functionality of project.
The game UI is clunky	Moderate	Unlikely	Make UI more simple
Game mechanics are bad due to lack of expirence	Moderate	Unlikely	Make the game mechanics simpler
Platform of development (Unity) stops supporting machine learning	Major	Rare	Find a different platform that supports Machine Learning
No more support for chosen Machine Learning library	Major	Rare	Find a different machine learning library