# Machine Learning Football Playing Agents

**Ryan Conway**

**Brian Doyle**

**James Kinsella**

B.Sc.(Hons) in Software Development

April 26, 2019

**Final Year Project**

Advised by: Daniel Craig and Gerard Harrison

Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

# Contents

# List of Figures

# About this project

**Abstract**
Artificial Intelligence has been the go-to buzzword in technology circles for decades, with proponents claiming it to be the panacea for our modern age. As such, why not apply the concept of machine learning to a more mundane application?

The objective of this project was to to see if we could train agents through machine learning in a Unity environment, while passing data to and from the application externally using Jupyter Notebooks, and to achieve a particular task, i.e play football.

The ultimate goal of our Final Year project is to create a three dimensional environment akin to a football pitch, in which two teams of agents compete to score goals for their respective teams. The simulation ends when a team scores a predetermined number of goals. These agents should be controlled by a Neural Network, with the capability to recognise their own team members, members of the opposition team, their target (the football) and their objective (the opposition goal).

**Authors**
The authors of this project are Brian Doyle, Ryan Conway and James Kinsella, who are three fourth year students studying for a Bachelors of Science Honours Degree in Computing in Software Development in the GMIT Dublin Road campus.

**Acknowledgements**

The authors would like to thank our supervisors Gerard Harrison and Daniel Cregg for the time and effort they put into helping us with our project, as well as all of the tips and troubleshooting assistance they supplied.

We would also like to thank the lecturers and faculty of the GMIT Dublin Road campus, in particular Dr. John Healy, whom we found to be a constant source of useful information on programming practices, life in industry and linking the nuances of technology and modern life to ancient Greek parables.

# Chapter 1

# Introduction

## 1.1 Overview

This chapter will outline the context and scope of the project. It will explain the objectives we had for the project and the specifications for the software, as well as detail our initial ideas and thoughts behind the design and it's implementation.

## 1.2 Idea

At the beginning of our 4th year in Software Development, we were tasked with coming up with an idea that would be suited to a level 8 final year project, so we knew we would have to design something that combined multiple different technologies and languages, and also challenged our own knowledge thus far. Our decision on what we were going to design sparked into life while we were exploring different possibilities of what to develop and what platform and languages we could work with. We stumbled upon a video from an event called the "Robocup", where teams of Artificially Intelligent Robots would play a game of soccer. After seeing this and being inspired, we eventually settled on the idea to develop some form of AI controlled agent. We had no experience with physical hardware like the "Robocup" utilised, and we wanted to find an environment that would suit what we aimed to do. For this reason, we decided to used the Unity game engine environment to act as the base for our AI controlled agents. We had the option to used Unity in a previous years module, however we had never really delved into the environment in much detail, and thus we still had a lot to learn about Unity's inner workings and plugins.

In order to control our AI agents, we would need to design a neural network somehow. We knew we were going to have a module in AI during our second semester, however at the time we had never touched anything to do with AI or neural networks, and our research into the field seemed to point us towards using python. Python seemed to be perfect to design the machine learning side of our project, as the language has a vast array of open source libraries free for us to use.

In the end we finished up our research and design with an idea to create a 3D Unity environment of a football stadium, with AI controlled agents playing football logically, all controlled by a python neural network externally to the Unity environment.

## 1.3 The Application

This application is a simulation of agents moving independently in a 3D through a neural network developed using python. We plan to have these agents recognise the object around them in the environment and use these to their advantage. With this we hope to add some personalities to each of these agents so that they make decisions based on what position they are in, and what to prioritise when in certain situations. We also hope to implement a win condition so that the simulation stops after a certain amount of time has passed or a certain amount of goals have been scored by a single team or agent.

## 1.4 Scope

As this is a final year software development project, we felt that this idea would provide many learning outcomes associated with the scope at this level. We were encouraged to use technologies that were new to us so with little to no experience in both unity and python we felt our idea slotted right into this category. As the project developed further we dove into other possible technologies that would benefit our development and working environment.

The purpose of this application is to train and reinforce the actions of particular agents on opposing teams, so that they work together to achieve an objective. In this case that objective is to score goals until a determined limit is reached. The scope of this project incorporates many technologies, such as Unity, ML-Agents Toolkit, Jupyter Notebooks and Python.

This project intends to show the applications of reinforced learning algorithms in a neural network, how they are applied within and the utility of such learning algorithms in more mundane day to day applications.

## 1.5 Objectives

The objectives of this project are:

- Design a working 3D environment in Unity to best suit our needs.
- Find a way to connect Unity and python together, allowing for data to be transferred between both the environment and the external code.
- Pass movements into the Unity environment using the external code.

- Pull observations from the Unity environment back into the external code.
- Control the agents using the external neural network.

## 1.6 Chapter Summaries

### 1.6.1 Introduction

This chapter contains the context for the entire project covering where the idea came from, what the project is about, what the objectives are for the project going forward, and the location and different elements of our GitHub Repository.

### 1.6.2 Methodology

This chapter describes the way the project was approached and managed. It also gives a description of how the project was researched and developed.

### 1.6.3 Technology

This chapter discussed the technologies that were researched and used in our project. It gives an insight into why the technologies used were chosen and what alternatives could have been used.

### 1.6.4 System Design

This chapter gives an explanation of how the entire system architecture was designed and how it all connects together. Diagrams are provided to further explain each individual element of the system.

### 1.6.5 System Evaluation

This chapter evaluates the project and the progress that was made towards it's completion, and highlights where the project could have been improved upon. It discussed the problems faced and the impact these had on the development of the project.

### 1.6.6 Conclusion

The conclusion gives a summary of our findings, outcomes and experiences we all had during the development of this project.

## 1.7 GitHub Repository

The GitHub repository for this project can be found at:
https://github.com/BrianD147/4th-Year-Project.

The sections below describe the different components in the repository and a link to each part

### 1.7.1 README

This contains a brief introduction and description of each component of the system. It also contains an installation guide as to how to install and run the project.

### 1.7.2 4th-Year-Project

This contains the main Unity environment code used to design the 3d stadium environment and the agents. Also contains the builds necessary to work with the notebooks within this project.

### 1.7.3 Project Dissertation

This contains a latex project used to write and develop out project dissertation and a PDF of the complete dissertation.

### 1.7.4 ML-Agents

This contains the ml agents python package, which is part of the ML-Agents Toolkit. This is a Python API that allows direct interaction with the Unity game engine as well as a collection of trainers and algorithms to train agents in the Unity environment.

### 1.7.5 Notebooks

This contains Jupyter notebooks used to run the python scripts and connect them to the Unity environment, with help from the ml-agents Python API. This folder contains 4 notebooks that each run different simulation types, and another notebook with an example of a Keras Neural Network.

### 1.7.6 Group Project 2018

This contains an initial document stating our initial goal and a road map we wished to adhere by as development progressed.

### 1.7.7 IronPython VS ML-Agents

This contains a document comparing the IronPython plugin with ML-Agents and discussing the advantages ML-Agents has over Iron-Python, concluding in our reasons for choosing ML-Agents going forward.

### 1.7.8 Input Data

This contains a file stating the input data and attributes intended to be passed into the neural network, and the set of outputs the network would output. Along with template data arrays for inputs and outputs.

### 1.7.9 Setup for Player Controller

This contains a brief explanation of how user controlled movement can be incorporated into the ML-Agents Academy and Brain prefabs.

# Chapter 2

# Methodology

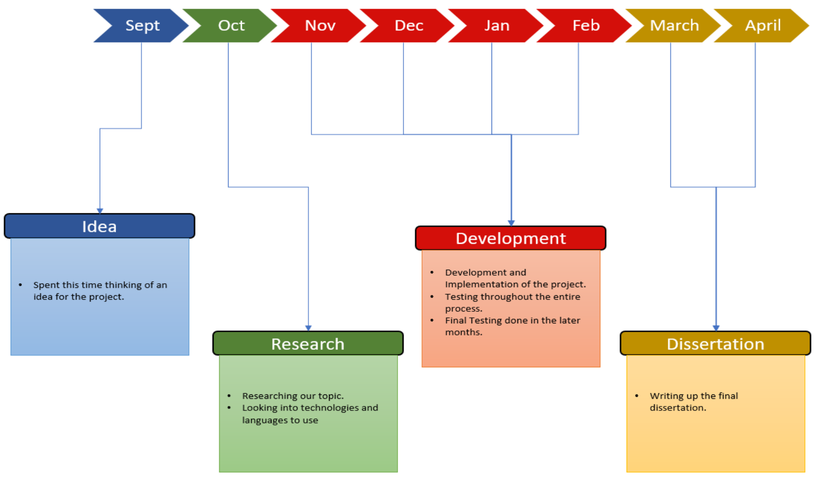

Sept
Oct
Nov
Dec
Jan
Feb
March
April
Idea
Spent this time thinking of an idea for the project.
Research
Researching our topic.
Looking into technologies and languages to use
Development
Development and Implementation of the project.
Testing throughout the entire process.
Final Testing done in the later months.
Dissertation
Writing up the final dissertation.

Figure 2.1: Project Timeline

## 2.1 Overview

In this section we will review our approach to the project development, describe the methodology used and how it was implemented and how the development of the project progressed overall. At the conclusion of this methodology the reader should be left with a good sense of the project scale, as well as the steps taken throughout.
For this project we wanted to use an iterative design process, so

changes and issues could easily be kept track of and dealt with in an easy and efficient manner. As such we felt the Agile method to be the best fit for us, as one of the team members has had previous experience in industry during a summer internship. His first hand experience helped guide us throughout the project, and gave the rest of us a feel on how it may be like to work in industry in the future.
The Agile method is a particular approach to project management, which is ubiquitous in the environment of Software Development. This method assists teams in responding to the unpredictability of constructing software, such as bugs in code, changing requirements from the client, etc. Agile uses incremental/iterative work sequences commonly known as Sprints, which is a period of time allocated for a particular phase of a project. Time is a precious resource when developing software, and as such a Sprint is considered complete once the time period expires. While some objectives set out for the Sprint may not have been realised, these additions will need to be put on the back burner in order to keep up with the Agile development cycle.
While we liked the approach the Agile method allowed us to use, we did not adhere to it in the strictest of senses. For instance, we did not appoint someone to the role of Scrum Master, preferring a more group oriented approach to problem solving and troubleshooting. A Scrum Master would be more suited to an industry environment, where they could be used as a point of contact or cohesion between large teams with differing responsibilities.
Another aspect of the Agile Methodology we utilised were ”Stand ups”. This is the term used for a daily progress meeting, traditionally held within a development area. These meetings are a quick way to gauge progress on individual aspects of a project, and are a great way to keep everyone informed and motivated. (once or twice a week for stand ups, library room, discuss work individually. Well informed anyways)

Some general principles of Agile include:

- Satisfy the client and continually develop software
- Changes in requirements are embraced
- Frequent delivery of working software
- Close working relationship between client and developer(s)
- Face to face communication is integral to cohesive development, and working relations as a whole
- Developers and clients should maintain a good relationship for the benefit of the project
- Working software is the primary measurement of progress
- At regular intervals, the team will reflect on how to become more effective, and they will tune and adjust their behavior accordingly

## 2.2 Sprint 1: Idea

Our first sprint was used to spitball potential ideas for a project, including what problem could we tackle, relevant technologies we could apply to said problem and in what manner we could approach it.

A particular point we needed to take note of was the scope of the project. The project would need to encompass a number of technologies that were not only relevant in relation to our course or module, but also technologies that are being used in day to day applications.

As noted in our ”Ideas” subsection, we stumbled across a video from an event called the ”Robocup”, in which Artificially Intelligent robots competed in a game of football. Finding this video was the seminal point in our project, where we collectively agreed that agents controlled by Artificial Intelligence was a valid route to go down. With our idea firmly in place it was time to move onto the research phase, in which we searched for relevant platforms, technologies and languages.

## 2.3 Sprint 2: Research

The next Sprint focused on researching our materials for the project, such as a suitable environment we could use, what languages we could utilize and if there were any external libraries or resources we could use to improve the overall scope.

The types of research undertaken in this sprint ranged from articles, videos, Stack Overflow forum posts and peer-to-peer, asking colleagues about their previous experiences with certain technologies. We also reviewed many technologically related papers through the use of Google Scholar and the GMIT Library site.

For our environment we settled on the Unity game engine, in which we had some previous experience in a different module. This environment allowed us to create a 3D framework for the agents, so progress could be visualized as well as documented in a traditional way. Unity also makes use of the C# language, which allowed us to build the framework in the object-orientated style of which we had grown accustomed. For our language of choice we settled on Python, due to the apparent ubiquity of the language in machine learning and data modelling circles, as well as the comprehensive standard library and open source libraries it offered.

The ML-Agent Toolkit is an open-source Unity plugin that enables games and simulations to serve as environments for training intelligent agents. We felt the addition of the Toolkit would take much of the hassle out of creating a machine learning framework ourselves.To still make use of the Python language itself we made used Jupyter Notebooks to pass information to and from unity/ML-Agents. We also had previous experience with Jupyter Notebooks from another module, so felt comfortable working with this program. More information on this subject can be found later, in the

”Technology” section.With the basics gathered it was time to move onto initial development.

## 2.4 Sprint 3: Initial Development

Set up environment, GitHub, etc

Once we had established a framework we were happy with work finally began on the foundations of the project. To tackle the problem of version control we utilized GitHub, which we have had extensive experience using in the past. Group members were then added as collaborators, so we could work on the project in unison. Once the blank Unity environment was created it was pushed to the GitHub, as well as any relevant research materials we had to date.

The task of setting up the Jupyter Notebooks, Unity and ML-Agents fell to Ryan, Brian and James respectively.

Documentation on ML-Agents was scoured over in order to integrate it into Unity, which required the use of the Jupyter Notebooks themselves. It was at this stage the team understood how interconnected these individual processes needed to be in order for the project to operate efficiently.

An environment in Unity was created for our initial testing purposes, namely a small football pitch with translucent walls and a ceiling, so any chances of assets ”escaping” the testing area were minimized. This environment also contained the two goals that would be used by the opposing teams. Next came the creation of other assets, namely the ball itself and the first agents. By now the project seemed to be taking shape. Next, we move onto integration and testing.

During this sprint we started having code review session. These sessions would be held at the end of every week during the sprint. For this we booked a room in the library and reviewed each others code in-dept and made comments on what could be improved upon and what they could start working on next.

One issue encountered during this sprint was in relation to ML-Agents, and how GitHub handles large files. This caused a blocking issue within the sprint, which meant all of the members of our team could not progress until this issue was resolved.

This issue was eventually resolved by adding the necessary folder to a public google drive address that anyone can access. The wiki on the GitHub page explains how to implement this folder into the project and contains a link to the google drive where it is stored.

## 2.5 Sprint 4: Integration and Testing

Integration testing of ML-Agents, Unity, Notebooks, etc

The next sprint focused on the integration and testing of the various technologies and platforms we had chosen.

The application works by using Unity to ”host” the ML-Agents toolkit, as it is essentially a Unity add-on in itself. Data within the ML-Agents Toolkit is passed to and from the Jupyter notebooks. The Notebook is used as a connector, staging blocks of code to execute in sequence so Unity doesn't become overloaded, as well as ease of debugging later.

After some time testing and debugging we finally had all three individual components integrated into one cohesive application. During this time we passed random values to the agents in the environment, in order to test the robustness of the Jupyter Notebooks in passing values, and to see how the agents reacted in the environment.

Another blocking issue arose during this sprint which saw the ML-Agents package receive a version update. This update brought a complete rework to the system and how all its components were implemented, including a change to the brains and academy which caused a big issue for us. This issue brought up the question of what to do going forward, were we to update to the newest version and re-write all we had up until now? Or would we stick to the previous version and continue to develop on that?

In the end we decided against the update as it brought too many new changes in that would render all the work we had put in up until now unusable.

## 2.6 Sprint 5: Functionality and Testing

This Sprint focused on the crux of the project, controlling agents in a meaningful manner using Artificial Intelligence, and also where we encountered our issue. These issues can be read about in detail below, in the ”Conclusion” section.

During this time we focused on a final code review, adding elements that we felt were necessary in order to explain our project and its intentions, or adding more to our current code base and design in order to provide clarity. Time was also spent making note of areas in which our project or write up may not have been up to scratch, and added any issues to the workload for the final sprint.

## 2.7 Sprint 6: Cleanup & Dissertation

The final Sprint was focused on project cleanup, as well as work on our accompanying minor dissertation. This time was spent implementing our research into the dissertation, as well as relevant images, graphs, blurbs and context to our approach.

## 2.8 Testing

### 2.8.1 Unit Testing

Unit Testing is a level of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software performs as designed [1]. We applied unit testing in the early stages of development when it came to testing the scripts we wrote for objects in our Unity scenes, such as moving the player by assessing how they moved with certain values, then changing those values to which suited best.

### 2.8.2 System Testing

System Testing is a level of software testing where a complete and integrated software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements [2]. We applied system testing in the closing stages of the Unity development of this project as we felt we had all the necessary pieces relevant for this area of the project. We tested how well the agent moved in the environment, how it worked with the ball, how it worked with the surrounding walls, the goals and the opposition.

### 2.8.3 Regression Testing

Regression Testing is a type of software testing that intends to ensure that changes (enhancements or defect fixes) to the software have not adversely affected it [3] [4]. This testing was applied in the latter stages of the project, when the Unity environment was ready and we could run the simulation directly from the notebook. This meant that we could make small changes to the notebook without it impacting the Unity environment as a whole.

# Chapter 3

# Technology Review

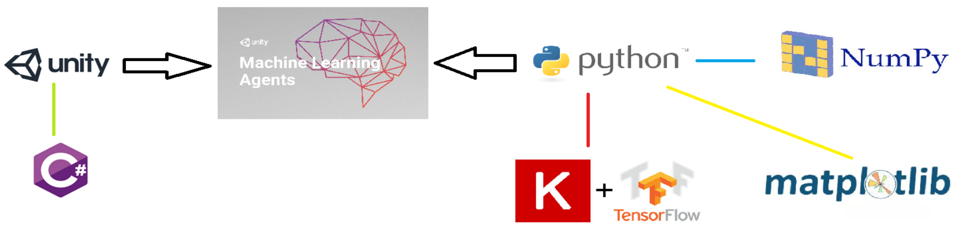


Figure 3.1: Technologies used

## 3.1 History of AI

The initial idea of artificial intelligence was first brought around during the second world war where Alan Turing and his team developed the Bombe machine, a device capable of deciphering messages from the enigma machine. Turing stated that a machine that could converse with a human, without the human knowing that they were conversing with a machine could be said to be "Intelligent". With this, the foundations for artificial intelligence was created.
John McCarthy is said to be the "father" of Artificial Intelligence, coining the term in the year 1955. He invented his own programming language, LISP which became the programming language of choice for AI development [5].
Despite AI development being around for so long, research fell sub-

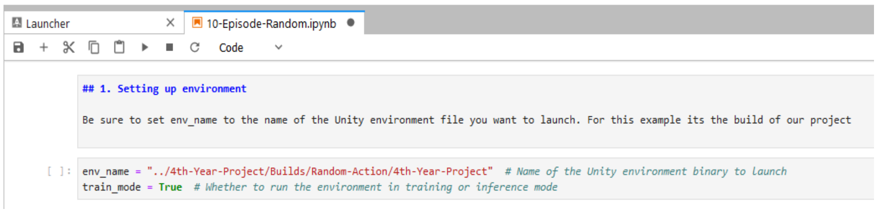
ject to what was known as “AI Winters”. During these phases funding for development of AI was reduced due to lack of processing power and interest fell short. With the turn of the century, research started to pick up again due to the advancements in hardware and the processing power of machines.

## 3.2 Machine Learning

Machine learning (ML) is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves [6] [7].
One of the current trends in modern day software development is the exploration into the world of Artificial intelligence. With the recent boom of Artificial Intelligence and Machine learning, we thought it would be a good idea to jump onto the bandwagon and see what it was all about [8]. In this paper we will describe in detail our process of creating an unity environment with the goal of teaching an agent how to play soccer, the challenges encountered throughout and the research done into the areas of Machine Learning and the Unity environment. We will then discuss how we implemented this idea with ideas acquired from our research. We will then discuss our results, what we learned and how we could possible improve on in the future with further research.

### 3.2.1 How it works

The notebooks are broken up into individual cells. Each individual cell can be specified into three different types of syntax: Markdown, Python and Raw text. Below is an example of both a Markdown and a python cell taken from our project before they are executed.



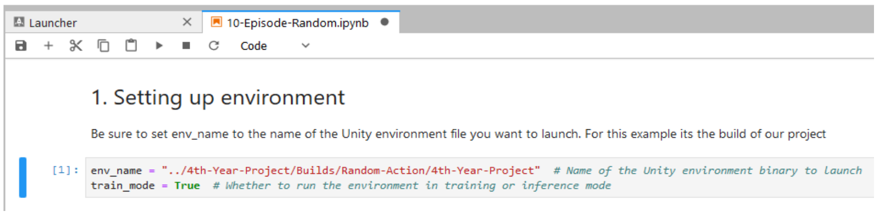
```
## 1. Setting up environment

Be sure to set env_name to the name of the Unity environment file you want to launch. For this example its the build of our project

[ ]: env_name = "../4th-Year-Project/Builds/Random-Action/4th-Year-Project"  # Name of the Unity environment binary to launch
     train_mode = True  # Whether to run the environment in training or inference mode
```

Figure 3.2: Notebook cells before execution

As you can see the first cell block contains the markdown syntax and the second cell containing python syntax before they have been executed.



1. Setting up environment

Be sure to set env_name to the name of the Unity environment file you want to launch. For this example its the build of our project

```
[1]: env_name = "../4th-Year-Project/Builds/Random-Action/4th-Year-Project"  # Name of the Unity environment binary to launch
     train_mode = True  # Whether to run the environment in training or inference mode
```

Figure 3.3: Notebook cells after execution

In this image we see both cells after being executed. The number ”1” to the left of the cell indicates that this has finished running and its the first python cell that has been executed in the notebook.

## 3.3 Technologies Review

This chapter will discuss in detail the technologies that we utilised in the project. It will also provide examples of the research we did into the different types of possible technologies. It will also highlight the rationale behind the decisions we made to use certain technologies over others.

## 3.4 Game Engines

In this section, we will discuss the possible game engines we can use to run our environment in. We don’t want to try use actual hardware and would prefer to run our environment in a game engine.

### 3.4.1 Unity

Unity is a real-time game engine developed by Unity Technologies and released back in June of 2005. Originally designed as an OSX-exclusive game engine, the software has been developed upon and extended to support a host of other platforms also [9]. Unity has made a name for itself, making game development accessible to everyone, as shown by the sheer amount of indie game developers using Unity as there main game engine tool.

#### Why use Unity

The main pull Unity has over other game engines is it's ease of use, the software has been designed to make it easy to design games from the ground up, with easy to understand UI and an overall aim to create functional, smart, and sometimes simple games, rather than complex games with high quality graphics. Also the fact that the software is free to download and use is a big plus [10].

### 3.4.2 Unreal Engine

Unreal Engine was developed alongside it's debut title ”Unreal” in 1998 by Epic Games. The game engine has kept a reputation of it being a high end piece of software, both due to it's price plans, and it's ability to generate incredible graphics [11].

#### Why use Unreal

The software has had some problems in the past with it's steep learning curve. Certainly not a beginners piece of software, however if someone who has had previous experience with coding and other 3d programs is looking to create a highly detailed expansive game, they need look no further than Unreal Engine [12].

### 3.4.3 Our decision

We decided to use Unity over Unreal Engine, firstly because of the learning curve. We had the opportunity to use Unity previously in a module, so we all had a slight bit of experience. Also the fact Unity is completely free, whereas Unreal Engine requires a payment plan to be set up, was what finally sold us on using Unity.

## 3.5 Neural Network Languages

In this section we will discuss the different languages we reviewed to write our neural network in. The following programming languages have all been noted as strong languages for neural network design.

### 3.5.1 Python

Created by Guido van Rossum, Python is a high level, general purpose programming language first released in 1991. Pythons design philosophy emphasizes code readability, meaning the language utilises a significant amount of white space [13]. The language supports multiple programming paradigms, so programs written procedurally will work just as well as an object oriented program design [6].

#### Why Python

Simplicity is a major factor when talking about neural network design, so having a language that's easy to understand, as well as the fact that python has an extremely large library of useful frameworks to utilise makes it a very strong choice [14].

### 3.5.2 R - Programming language

R is a programming language and software environment used for statistical computing, and is great for producing informative graphics and data mining [15]. Originally introduced back in 1993, the language has stuck around due to it's presence as a truly general purpose programming language [7].

#### Why R

Although R doesn't originally appear to be a language that would suit neural network design, the sheer amount of libraries that R boasts, including RODBC, Gmodels, and Tm, which are all used in the field of machine learning, means the language actually has a strong case for being the correct one to use [16].

### 3.5.3 Lisp

Lisp is a family of programming languages, originally specified way back in 1958. It is the second oldest high level programming language that is still in widespread use today. John McCarthy, the inventor of the language, is actually known as the father of Artificial Intelligence, and many of the languages features have been implemented and migrated over into more modern programming languages [17].

#### Why Lisp

Apart from it's inventor holding the title of the father of AI, Lisp actually became the favoured language for AI development and research shortly after it's introduction [18].

### 3.5.4 Our decision

We decided to use Python as our language to build the neural network in due to the fact that the language is currently considered to be on top when it comes to AI development. Even though R and Lisp both were strong candidates for what language to use, the fact that Python is designed to be so simplistic and easy to understand made it the best choice for us, as we would need to be able to get our heads around the inner workings of neural networks quickly.

## 3.6 Connector Software

In this section we will discuss the possible ways to connect the neural network code to the game engine environment.

### 3.6.1 Jupyter Notebooks

The Jupyter Notebook App is a server-client application that allows the editing and running of notebook documents via a web browser [19]. Notebook documents are produced by the App and can contain both computer code as well as rich text elements. These documents acts as an executable which can run whatever code they consist of [20].

#### Why Jupyter Notebooks

We have had previous experience with notebooks, using them in a previous module, so we already have an understanding of their inner workings. We also found plenty of examples of other people connecting game engine environments to external code. Having a previous understanding of the ins and outs of these notebook documents will be a great advantage to using notebooks as a connector between the environment and external code [21].

### 3.6.2 Transmission Control Protocol connection

TCP is one of the main protocols of the Internet protocol suite [22]. TCP provides a reliable, ordered, and error-checked delivery of a stream of bytes between applications communicating via an IP network [23].

#### Why Transmission Control Protocol

A TCP connection has been attempted to connect a game engine environment to external neural network code by others [24]. We found some examples of other attempts, with some positive results. The bonus to using a TCP connection would be the fact that literally anything could be passed between the environment and the external code. Json objects could be passed back and forth meaning any input needed for the neural network, and subsequent output, could all be passed back and forth with ease, once a connection between the two had been established.

#### Our Decision

We decided to use Jupyter Notebooks as our connection between the game engine environment and the external code since we have already had some experience with the application, as well as the fact that compaired to a TCP connection, using a notebook should be much simpler to initialise, since if we were to attempt a TCP connection, an IP network would have to be initialised to run the project on, which could lead to difficulties, and we would prefer to use something we know can work.

#### IronPython

IronPython is an open-source implementation of Python [25]. The implementation uses the .NET framework, as well as the entire suite Python libraries. The class library also contains language interoperability, meaning each language can use code written in another language.

#### Why IronPython

IronPython having the ability to use the .NET framework could work well for us connecting our environment to our external code. The .NET framework's expansive class library has plenty of useful resources that we could utilise [26].

### 3.6.3 ML-Agents

ML-Agents is a new open-source Unity plugin designed by Unity themselves, which aims to provide an expansive toolkit within the

Unity environment to help with the development of machine learning [27]. The software enables games and simulations to serve as environments for training AI agents.

#### Why ML-Agents

Using ML-Agents is the perfect choice for us, since we have decided to use the Unity engine to develop our 3d environment in. The documentation for ML-Agents shows us exactly how we could use this plugin to aid us in creating agents in our Unity environment.

### 3.6.4 Neural network libraries

In this section we will discuss the possible neural network libraries we could utilise in our project. Using a machine learning library could be very useful for developing our neural network and ML models, as well as offer easier debugging options for testing.

#### TensorFlow

TensorFlow is an end-to-end open source platform for machine learning. Its flexible ecosystem of tools, libraries and resources allow for easy development and deployment of ML powered applications [28].

#### Why Tensorflow

Tensorflow is a very well-known platform for ML powered applications. Having a platform that we can use to train models easily could be very useful when designing our own models. [29]

#### Sony's neural network libraries

Neural Network Libraries by Sony is an open source software aiming to make research, development and implementation of neural networks more efficient. [30]

#### Why Sony's neural network libraries

Having a fully optimised library of ML related code could be extremely useful when designing and implementing our own neural network. Although the software isn't as widespread as TensorFlow, it could be better suited for our needs.

#### Our Decision

We decided to have TenserFlow as our go to neural network library for two main reasons. Firstly, Sony's neural network libraries aren't as widely used as TenserFlow, meaning examples and other documentation is harder to come by. And secondly, we started a module in our course which utilises TensorFlow, meaning we are already starting to learn what the libraries can be used for and how we could adapt it into our own project if needed.

### 3.6.5 Unity Test Tools

Unity Test Tools is a test framework for any games and interactive content made in Unity. [31] The framework is built to feel like a natural extension of the Unity Editor, so everything should be intuitive to anyone with experience using the Editor itself.

#### Why Unity Test Tools

The fact that these tools are easily accessible from the Unity asset store, as well as the fact they were designed by the Unity developers themselves, means they will work perfectly with the environment we build. [9]

#### Our decision

We decided the use of the Unity Test Tools would actually not be very useful, since the fact that the majority of the work being performed will be external to the Unity environment. Having test data from just the environment itself isn't necessary for the type of project we are aiming to create.

### 3.6.6 Docker

Docker is a tool designed to help create, deploy and run applications. The tool uses containers to package all the necessary components, including libraries and other dependencies, into one single package. [32] This means that any developer can rest assured the application will run on any machine.

#### Why Docker

Using Docker would allow us to focus on the coding and development of the project, without having to worry about the system that our application will ultimately be running on. Having everything packaged together would help with the final deployment of the project, guaranteeing the entire project would run on any system without a shadow of a doubt. [33]

#### Our Decision

In the end we decided not to work using Docker, even though the tool is very intriguing and would be a major help running the project in the later stages, through some research we found complaints that the containers can be subject to performance overhead, and with our project, running the simulations at a consistent rate is very important. Also our project consists of multiple possible scenes and scenarios to run from multiple different notebooks, meaning that using Docker would not allow the final user to run whichever scenario they wishes, as Docker uses a single command to run it's packages content, and does not offer the ability to run multiple different parts of its package independently for the final user.

### 3.6.7 Anaconda

Anaconda is a free and open-source distribution of Python and R programming languages for scientific computing, including machine learning applications. [34] The distribution aims to simplify package management and deployment, offering over 1400 data-science packages.

#### Why Anaconda

Anaconda would offer an easy and efficient way for us to gather the necessary packages for our machine learning needs. If we decide to use either Python or R as our programming language of choice, then Anaconda may be a great option for us. [35]

#### Our Decision

We decided to utilise Anaconda since we are using Python to build the neural network side to our project. The distribution includes matplotlib, numpy, and keras, which we aim to also use in our project, meaning Anaconda will be very helpful in installing these packages.

## 3.7 Neural Networks

Neural network, a computer program that operates in a manner inspired by the natural neural network in the brain. The objective of such artificial neural networks is to perform such cognitive functions as problem solving and machine learning. These papers bring up some very interesting points that will help us in developing our own neural network [36] [37] [38].

## 3.8 Unity

Unity is a cross-platform real-time engine developed by Unity Technologies. The engine can be used to create both three-dimensional and two-dimensional games as well as simulations for its many platforms.

## 3.9 Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance.

## 3.10 Jupyter Notebook

Notebooks Through the use of the Jupyter notebook, we could develop our AI through python while also documenting it in a nice style with the use of the Markdown option. This provided a clear user-friendly experience when running the python code through the notebook with in depth explanations of what our code does. The Jupyter notebook app is a server to client application that allows users to create notebook documents and run then via the web browser. This app can be run on a local desktop without the requirement of internet access through the command line. There is also a similar version of this web app called Jupyter Lab. This version of the web app grants the user a more IDE-like experience, allowing the user to have many tabs in the same window. With our development we went with using the Jupyter Lab version over the Jupyter Notebook version as we found it more useful due to the number of notebooks we would be developing and the GUI similar to an IDE made it more comfortable to work in.

To open the Jupyter notebook version of the web application, enter the following into the command line

```
Jupyter notebook
```

To open the Jupyter lab version of the web application, enter the following into the command line

```
Jupyter lab
```

# Chapter 4

# System Design

## 4.1 UML - Diagram

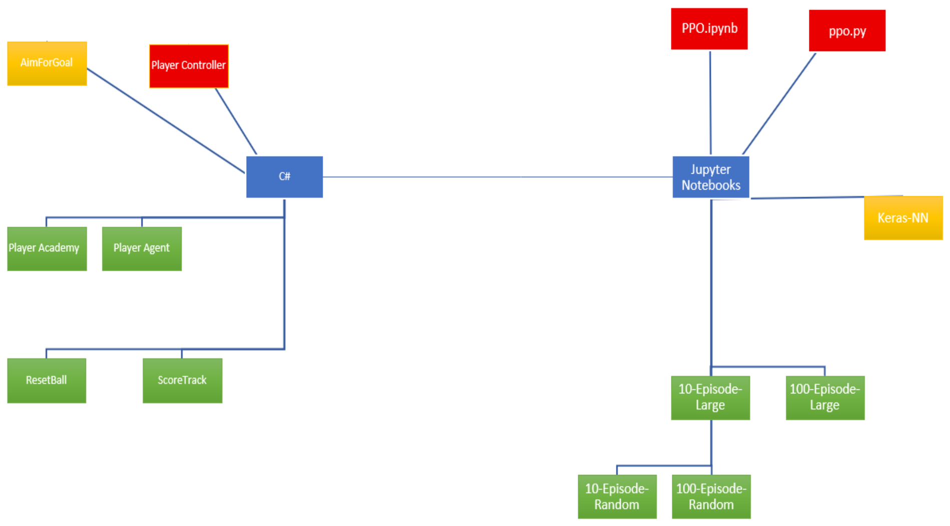


Figure 4.1: UML Diagram of our project files

Blue – Language/Technology used
Green – Files necessary for the project
Yellow – Not necessary but related to what we wanted to achieve
Red – Files relevant at the beginning of the project but less so now.

- PlayerAcademy - Script needed for the Academy Object in the Unity environment.
- PlayerAgent - Controls all the parameters handled by the player object in the Unity Environment, including the speed, kick power and position of the player.
- ResetBall - Handles the physics of the ball, resets its position whenever there is a goal or if the ball somehow leaves the stadium.
- ScoreTrack - Registers when the ball has entered the goal and increments the scoreboard accordingly.
- 10-Episode-Random - Contains the logic for running a simulation.
- 100-Episode-Random - Contains the logic for running a simulation.
- 10-Episode-Large - Contains the logic for running a simulation.
- 100-Episode-Large - Contains the logic for running a simulation.
- AimGoal - Script that pushes the players into better positions to score in the opposition goal - hard coded.
- Keras-NN - Example of a Keras Neural Network in action.
- PlayerController - Used at the start of the project to control the agent in the environment.
- PPO.ipynb - Proximal Policty Optimization notebook in the hopes of finding a way to develop the Neural Network.
- ppo.py - Python version of the above file.

## 4.2 Unity Environment

### 4.2.1 Unity - Scenes

#### Soccer-Rand-Enviro

In the “Soccer-Rand-Enviro” scene, a single agent is placed on the field. This scene is meant to offer a clear view of the random inputs notebook at work, leaving the entire area free for the agent to spawn into on each epoch run of the notebook. The scene originated as the starting scene we aimed to use to train the first agent in, and once we had a single agent working as we intended, we could move onto other scenes with more agents.

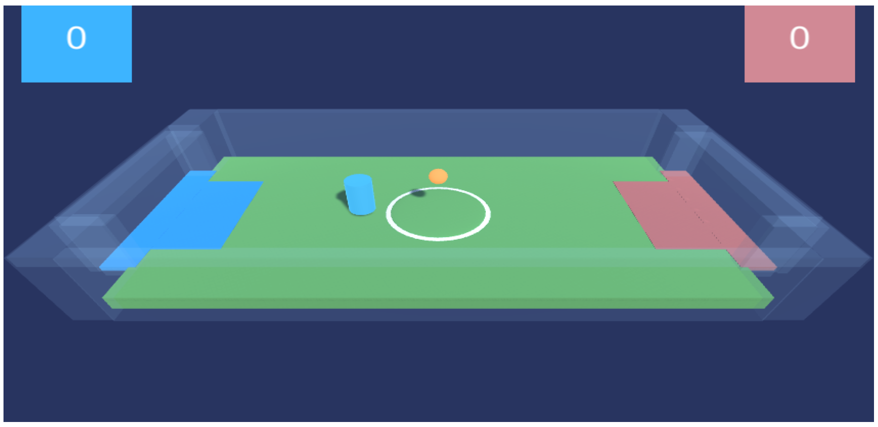


Figure 4.2: Scene with a single Agent

#### Soccer

In the “Soccer” scene, 6 agents are positioned on the field. This scene was meant to be the next step after getting a single agent working as intended. We were going to have multiple agents working with each other, and logically playing with teammates to attempt to score. Currently this scene acts as a showcase of what we wanted the project to perform like, with agents keeping there own positions on the pitch and attempting to score in the opposition’s goals.

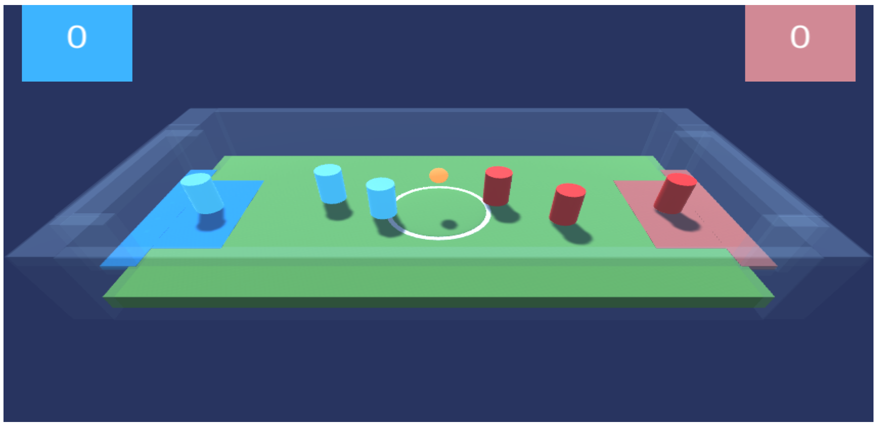


Figure 4.3: Scene with two teams, Each with a goalie, a defender and and attacker

**Soccer-2**

In the “Soccer-2” scene, more agents have been added to the pitch, making the total add up to 14 agents. This scene was going to offer us an environment to fully test the capabilities of our trained agents, allowing them to have multiple more teammate’s to incorporate into the input data and expand there complexity. Unfortunately we never got to use this scene as we wanted, and rather only ran the random input notebook with it, since we couldn’t pass the necessary input data back to the network.

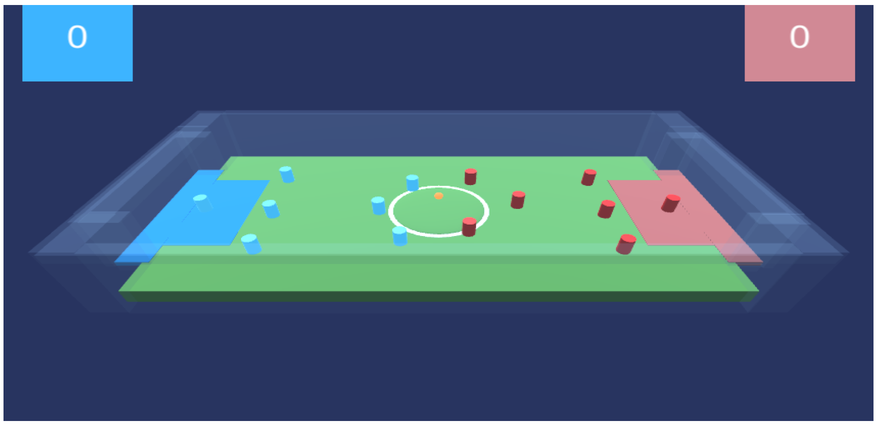


Figure 4.4: Larger scene with even more agents

### 4.2.2 Unity - Brain and Academy

#### Academy

An Academy orchestrates all the Agent and Brain objects in a Unity scene. Every scene containing Agents must contain a single Academy [39]. The object has many parameters that can be customized to suit the users needs, as can be seen here in fig 4.5. The Max Steps parameter indicate the total number of steps per-episode. 0 corresponds to episodes without a maximum number of steps. Once the step counter reaches maximum, the environment will reset [39]. The next parameter is the training configuration, where the user can set the width and height of the window, the Quality Level which controls the quality of the simulation, the Time Scale which controls the speed of which the simulation is run and the Target Frame Rate which indicates the FPS you wish the simulation to maintain.

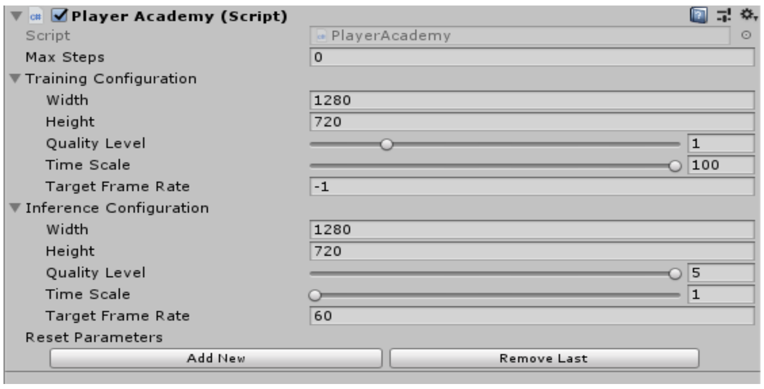
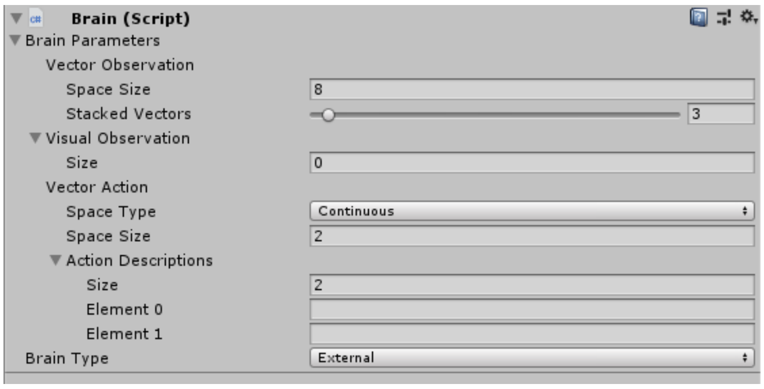

```
Player Academy (Script)
Script                   PlayerAcademy
Max Steps                0
Training Configuration
    Width                1280
    Height               720
    Quality Level        1
    Time Scale           100
    Target Frame Rate    -1
Inference Configuration
    Width                1280
    Height               720
    Quality Level        5
    Time Scale           1
    Target Frame Rate    60
Reset Parameters
```

Figure 4.5: Academy Parameters

#### Brain

The Brain encapsulates the decision making process. Every Agent must be assigned a Brain, but you can use the same Brain with more than one Agent. You can also create several Brains, attach each of the Brain to one or more than one Agent [40]. Here the main component we may want to change is the brain type, the option at the bottom of fig 4.6. As of ML-Agents version 0.5.0, there are 4 brain types: Player, Heuristic, Internal and External. The Player brain is used to map keyboard keys to Agent actions, which can be useful when testing the Agent code. The Heuristic brain is used to hand-code the Agent's logic by extending the Decision class. The Internal Brain is used to run an already trained brain or to train a brain internally. The External Brain is also used to run an already trained brain or to train a brain but when done externally (Via a notebook or any other external software).



| Brain (Script) | |
|---|---|
| Brain Parameters | |
| Vector Observation | |
| Space Size | 8 |
| Stacked Vectors | 3 |
| Visual Observation | |
| Size | 0 |
| Vector Action | |
| Space Type | Continuous |
| Space Size | 2 |
| Action Descriptions | |
| Size | 2 |
| Element 0 | |
| Element 1 | |
| Brain Type | External |

Figure 4.6: Brain Parameters

### 4.2.3 Unity - Logic

We aimed to have each player (agent) be capable of observing its own surroundings, so on each frame the agent would be able to make decisions on where it would suit best to move.
Each agent needed to be able to calculate distances relative to itself, such as the distance it is from the ball, the distance it is from its own goal, and the distance it is from the opposition goal as seen here in Fig 4.7.

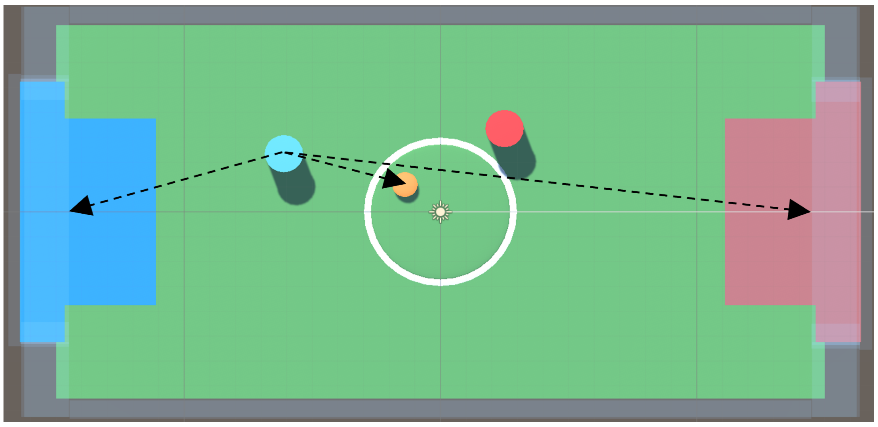


Figure 4.7: Distance of blue agent to each goal

Also the agent needed to be able to distancing information relating to the closest opponent to itself, such as the distance the closest opponent is to the ball, the distance the closest opponent is from the opposition goal, and the distance the closest opponent is from the agents own goal as seen here in Fig 4.8

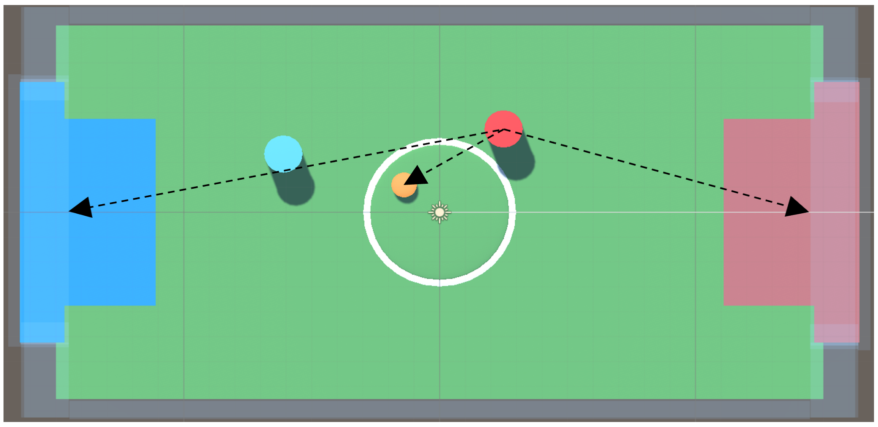


Figure 4.8: Distance of red agent to each goal

The overall position of the ball on the pitch would also have to be taken into account, so the agents must have been able to calculate the distance the ball is to the agent's own goal, as well as the distance the ball is to the opposition goal as seen in Fig 4.9.

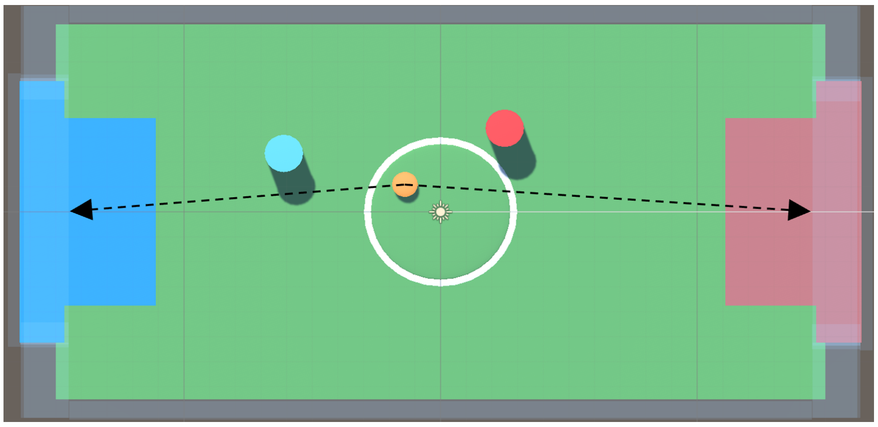


Figure 4.9: Distance of ball to each goal

Each agent needed to know what type of player they were, be it a goalkeeper, defender, or striker, and adjust its decision making accordingly, for example, in some scenarios it would be better for a goalkeeper to stay protecting the goals instead of rushing out to challenge the opponent with the ball, however a defender or striker might decide to push forward and challenge the ball in the same scenario, be it there is a goalkeeper positioned in their goals.
Along with each agent knowing which type of player they are, they also need to be able to detect what type of player the closest opposition is, as this could also affect the decision the agent makes, whether the agent should attempt to challenge the ball or if they are better off positioning themselves elsewhere.
Finally, the agent needs to be able to determine the angle the ball sits between themselves and the opposition goals, as well as the angle the ball sits between the agent and their own goal as seen in Fig 4.10. These angles need to be considered as they will also influence the agent's decision when moving frame by frame.

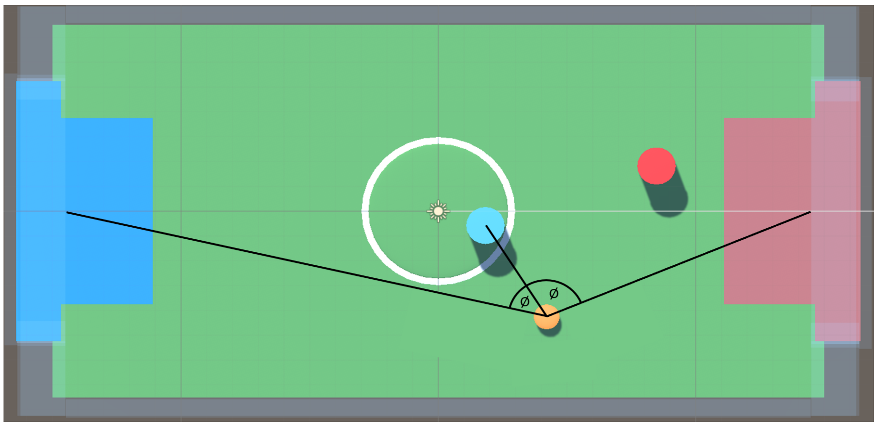


Figure 4.10: Angle of ball to goals and player

For example, it would be better for the goalkeeper to position themselves between the ball and the goal here, instead of simply rushing towards the ball and attempting to challenge the opponent as displayed in Fig 4.11.

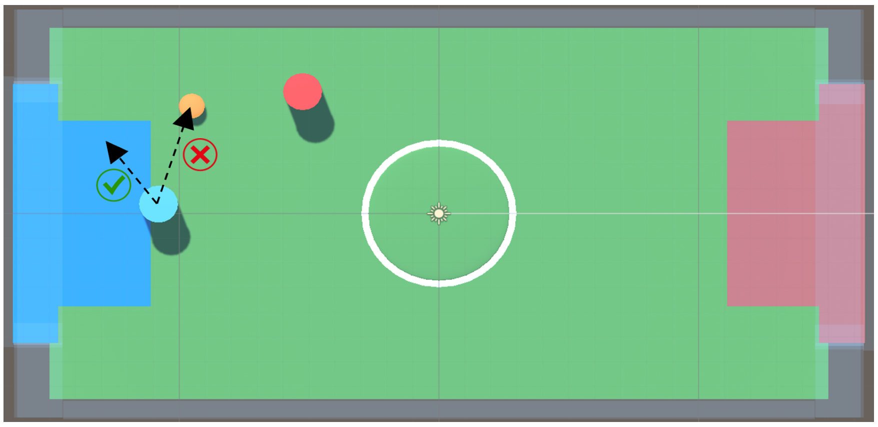


Figure 4.11: Ball prioritizing stopping goal over chasing ball

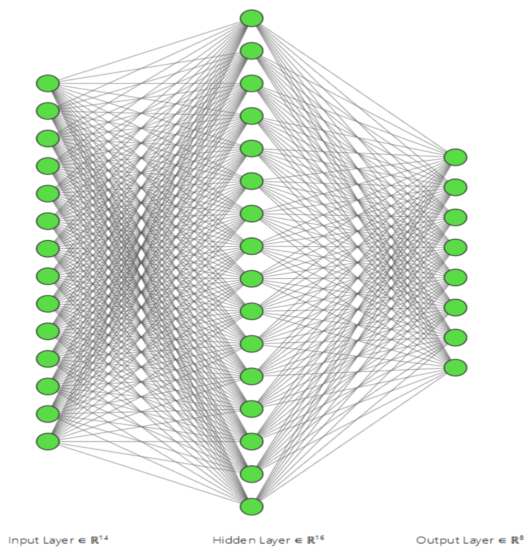

Input Layer $\in \mathbb{R}^{14}$
Hidden Layer $\in \mathbb{R}^{16}$
Output Layer $\in \mathbb{R}^{8}$

Figure 4.12: Neural Network Topology

## 4.3 Neural Network Design

The design for the neural network architecture, as seen in fig 4.12, consisted of 14 input nodes, a single hidden layer of 16 nodes, and 8 output nodes. The input nodes consist of multiple different value taken from the environment, including factors such as the type of agent being controlled, distances between the different agents on the pitch and the ball, the overall position of the ball on the pitch, as well as data pertaining to the angle the ball is between the agent and the target goals. The output nodes consist of each direction the agent could move at each episode. Meaning the neural network will use the data taken from the current state of the environment and the agent will move in the direction that it believes will end with the best outcome.

## 4.4 Notebook

### 4.4.1 How they work

These notebooks contains the logic required to run our simulations through different builds of our Unity environment. To do this it initially sets the the location of the build to a variable in the notebook. This variable is used later on in the notebook to sent and retrieve data from the Unity environment. We then imported all the necessary libraries such as Numpy, Matplotlib, Keras and the ML-Agents environment. This cell also checks what version of python the user has installed on their machine, as the ML-Agents package does not run on any other version than python 3. The next cell in the notebook accesses the environment and retrieves any brains that are present within that environment, and displays all information of that brain.This also launches the Unity launcher prompting the user to set their desired resolution and window size for the scene. Once the user presses play the scene will launch and stall. It is supposed to stall as its awaiting instructions from the notebook. The following cell resets the current environment and observes any observations in the scene. The next cell contains a for loop which iterates at the end of each episode. During each episode the cell accesses the brain within the environment and send random values to that agent. The user can see this simulation running if they navigate to the Unity window launched earlier. During each episode the cell accesses a rewards function located within the ”PlayerAgent” file of the Unity Environment. This function sends rewards to the notebook biased on the moves the agent takes in the environment, for example: If the agent was to hit the ball, positive values would be sent to the notebook, whereas if the agent had been moving for a while and not coming into contact with the ball negative values would be sent. Once each episode ended, the results would be accumulated and sorted in an array. Once all the episodes had ended, the environment would stall again, waiting for further instructions from the notebook. In the next cell the values that were sorted in the array were drawn onto a graph so we could visualize the progression or regression of the agent. The final cell in the notebook closes the environment as the simulation is complete.

#### 10-Episode-Random

Using the logic from above, this version of the notebook sets the episode counter to 10, and sends the Agent random inputs. The build used was the ”Random-Action” build which is located in the ”Builds” folder in the main project folder. This scene contains a single Agent and a single ball. The build path to this specific scene is located in the first cell of the notebook.

#### 100-Episode-Random

This notebook is similar to the previous notebook as it uses the same build, the only difference being that instead of 10 episodes this notebook runs 100 episodes. With such a large number of episodes the scene runs for much longer, thus giving us more results.

#### 10-Episode-Large-Soccer

This notebook contains the same logic as the previous two, but uses a different build. This build is called ”Soccer-Large” and is located in the same ”Builds” as the previous two examples. With this build, there are a total of 14 agents in the scene, 7 for each team. Also contained in this build is a script that pushes the agents to go towards the ball and keeps agents within their designated positions. For example: the ”goalie” agent will stay on his line and not wander up the pitch, whereas the ”striker” agent stays at the top of the pitch in front of the oppositions goal.

**100-Episode-Large-Soccer**

Again, based on the previous notebook this one uses the same build and logic, only difference being that the episode counter is set to 100, and in turn giving us more output results.

# Chapter 5

# System Evaluation

## 5.1 Testing

As mentioned above in the technology review, after looking into the possible testing tools available to /unity, we decided that it was best to do the majority of our testing through our notebooks. When running the simulations through the notebook it provided us with enough details that if something was to go wrong in the unity environment the notebook would be able to easily identify the problem, making it easier to troubleshoot it ourselves. Testing was constant throughout the project as for if we made any changes to the unity environment, the project would have to be rebuilt and the notebooks reran. Also due to the fact the python code in our notebooks were written into separate cells, it made it clearer to identify problems earlier on in the simulation. We seen this as part of a regression testing technique, where the tester or developer reruns functional and non-functional tests to confirm that a recent program, or code change has not indirectly affected the existing features within the system. This allowed us to run certain cells in the form of a test case rather than to rerun the notebook as a whole, giving a clearer insight to which parts of the code would give us specific problems if they were to occur during development.

## 5.2 Notebook – Results

### 5.2.1 Keras-NN

We used the notebooks to handle the machine learning side of things when it came to training the agents. With this we had to research which libraries we would need in order to develop a neural network. After a lot of research, we decided on using the Keras, NumPy and Matplotlib open source libraries as these libraries best suited our objective.

With this decided, one of our team members developed a notebook which showed a Neural Network in action. For this example, we used the IMDB data set as it was already built into the Keras library. This data set contains around 25,000 movie reviews from the website IMBD, each labelled with a positive and negative review. Each review has been pre-processed and encoded as a sequence of word indexes. With this set we extract some samples, training data to test against the data set and training targets to achieve with that data.

In our example We've extracted 10,000 words from both the start and the end of the data set and assigned each to a testing and training variable respectively. With this set up, the model can be set up accordingly. We've set the model to Sequential as we wanted a linear stack of layers for this example. Once defined we can add our input, hidden and output layers to the model. The input layer is the layer where we pass in the input from the data set (The data we want to be tested). The hidden layer is used to transform the inputs received from the previous layer into a suitable value that the output layer can use. In turn, the output layer transforms the hidden layers activation's into whatever scale we wanted our output to be on.

With the model made we then compile and fit the model to whatever specifications necessary. Once we fit and compiled these results, we then had to set the data we want to use to train the model, the number of episodes per each training session (epochs), the batch size of how many values we wanted to test against the model and the validation data, which contained the desired output and what the training data will be tested against.

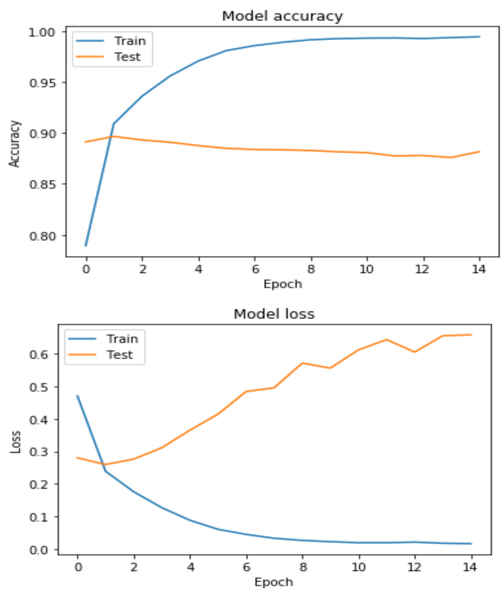


Figure 5.1: Graph of the Accuracy and Loss Percentage

The results shown in 5.1 indicate that the training set got more accurate as the number of episodes increased and eventually even surpassed the set it was trained against, thus showing the robustness and efficiency of the neural network.

### 5.2.2 10-Episode-Random

This notebooks results are based off of random inputs sent from the notebook to the Unity environment, and the rewards from that episode were then sent back to the notebook to be processed into positive or negative values. Positive values indicated that the agent was making the correct decisions, and the negative values indicated that the agent was making the wrong choices.

Due to the fact that the inputs to the agent were random, if the agent were to get more positive values in one episode they would not translate to the next episode, hence why the values in the graph in fig 5.2 look this way in comparison to the IMDB Neural Network results in fig 5.1.

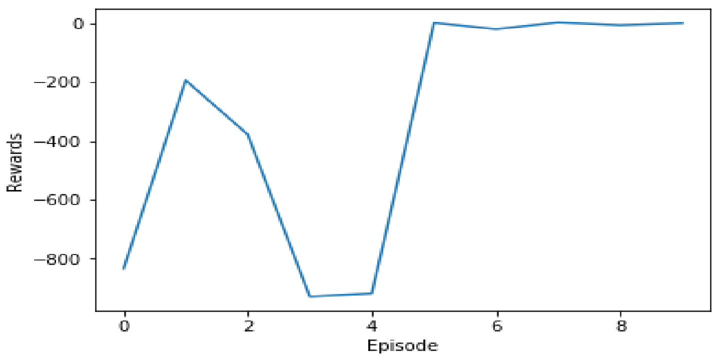


Figure 5.2: Graph of rewards from the Agent - 10 Episode Random

### 5.2.3 100-Episode-Random

For this notebook, we ran the same scene with the same scenario as the previous notebook "10-Episode-Random". The difference between this notebook and the last is that we ran the simulation with 100 episodes in comparison to the 10 in the previous notebook. Doing this meant our simulation ran for much longer, thus giving us more data to process. Again, this is due to random inputs from the notebook, therefore the results are skewed as seen in fig 5.3

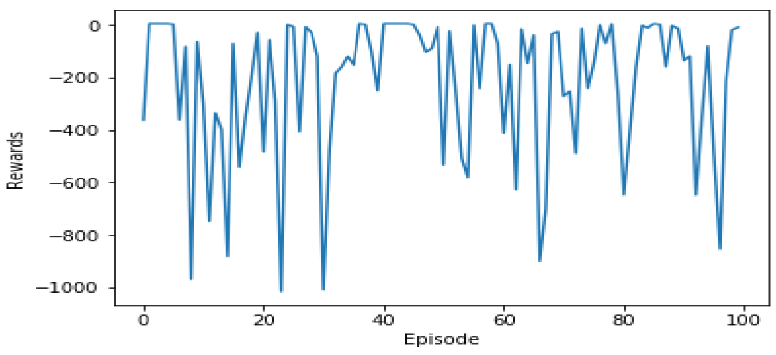


Figure 5.3: Graph of rewards from the Agent - 100 Episode Random

### 5.2.4 10-Episode-Large-Soccer

With this notebook we used a different scene to the previous two notebooks mentioned. This scene was much bigger and contained more agents within the scene. This scene represents what we intended to achieve with our project, in comparison to the previous scenes with the random input.

In this scene we've attached an extra C# script called "AimForGoal" This script contains logic that distinguishes which team the agent is on. Then script then deciphers whether the agent is a goalie, a defender or an attacker, and depending on the balls position on the pitch, the agent in the best position to move towards the ball. Each agent adjusts it position at the ball to aim towards the oppositions goal. Considering we could implement the Neural Network correctly into the agents, we wanted to showcase a more accurate representation of what it would look like if we had achieved this.

With the increase of agents and sizes, it also took a toll on the hardware of our machines when running this script. Due to this the episodes are longer, and the results accumulate slower, thus resulting in more drastic outputs. If we compare the results in fig 5.4 with the results from the 10-Episode-Random file in fig 5.2 we can see that the first set of results had slightly more consistency when it came to the end of the episode cycle, whereas the results from this set show that the rewards outputted has a drastic change in values for each episode that ran.

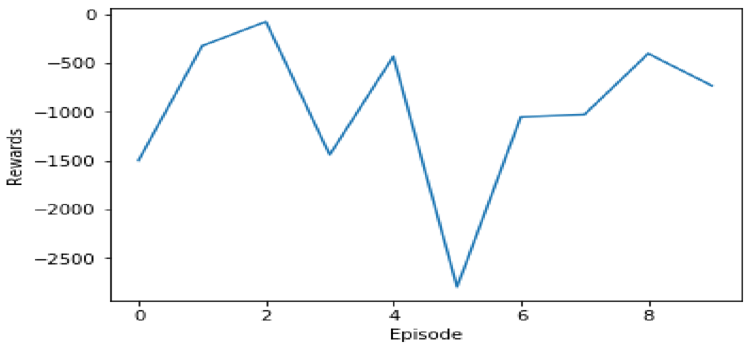


Figure 5.4: Graph of rewards from the Agent - Large 10 Episode Random

### 5.2.5 100-Episode-Large-Soccer

Similarly to the other 100 episode notebook, this is another notebook based on the previous example only running 100 episodes rather than 10. Again, with such a large scene and many agents in that scene, the simulation runs slower than all previous other examples. With the increase to the episode amount, also meant that we retrieved more data from the results. This data as seen here in fig 5.5 is more comparable to the 100-Episode-Random results in fig 5.3. We can see that the rewards took more of a dip during some episodes, possibly due to the fact that it ran much slower, thus adding or in this case subtracting more throughout the episode.

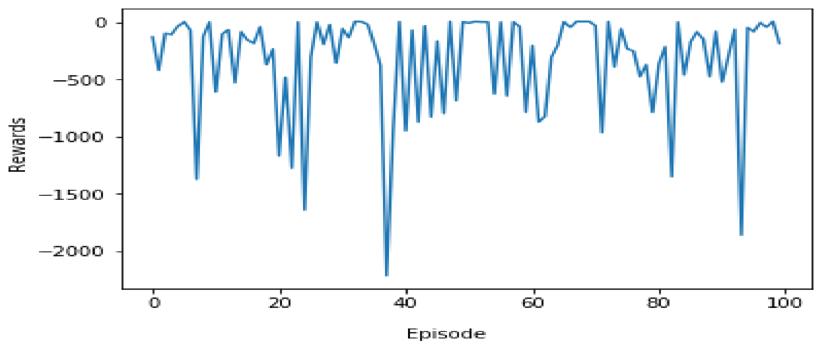


Figure 5.5: Graph of rewards from the Agent - Large 100 Episode Random

# Chapter 6

# Conclusion

## 6.1 Conclusion

This chapter will summarise the project in terms of the objectives of the initial proposal and will discuss the findings and outcomes of the project now it has concluded.

This project proposed the development of a 3d environment build in Unity consisting of multiple AI controlled agents playing football. External python code was proposed to run the neural network controlling the agents external to the Unity environment.

## 6.2 Goals and Objectives

The following shows our original objectives for the project, whether we managed to fulfil said objective, and a short comment on each objective, either how we managed to succeed or why we failed.

- [PASS] - Design a working 3d environment in Unity to best suit our needs. Our aim to design a working 3D Unity environment to hold our agents in was successful, with a football pitch surrounded by clear walls working perfectly to house the AI agents and ball.
- [PASS] - Find a way to connect Unity and python together, allowing for data to be transferred between both the environment

and the external code. We managed to connect the Unity environment to external python scripts using Jupyter Notebooks, acting as a buffer in between both and providing a clean and easy to understand interface for running the code.

- [PASS] - Pass movements into the Unity environment using the external code. We successfully passed random actions from the Jupyter notebooks into the Unity environment, moving the agents to random positions on the field on each frame.
- [FAIL] - Pull observations from the Unity environment back into the external code. We failed to find a way of passing any observations from the Unity environment back into the Jupyter notebooks, and thus into any python scripts.
- [FAIL] - Control the agents using the external neural network. We were unable to control any agents from an external neural network due to the previous objective failure. Having no observations to supply as input to any neural network meant controlling the agents wasn't possible.

## 6.3 Retrospective of the project

The following are the findings and outcomes from this project:

- A better decision could have been made in terms of using ML-Agents, since the plugin is actually still in early beta.
- ML-Agents went from version 0.5 to version 0.6 in the midst of our development, and with it many aspects of how the plugin worked previously changed, this meant that all the documentation on the previous version was also changed.
- We made the decision to stick to version 0.5, but now looking back perhaps it would have been more beneficial to upgrade and adapt the project to the newer 0.6.

- The documentation for ML-Agents wasn't actually as helpful as we originally thought, lots of pre-built examples exist, however an actual breakdown of what went into these examples does not.
- For any future development we may be involved in, versioning has clearly stated itself as being of utmost importance, for example, ML-Agents version 0.5 will only work with python 3, something that we were never made aware of in the ML-Agents documentation (ML-Agents 0.5 will not run with the latest version of ML-Agents, meaning ML-Agents 0.6 will not run any project developed in 0.5)
- Hardware was an issue running the Unity environment after intense training simulations had taken place.
- Scrum Agile Methodology provided good structure to our development process, however mid way through development, with so many setbacks and subsequent blocking issues, the sprint schedule was effected in the later stages, messing with our overall time management.
- Despite the problems with using the Scrum Agile Methodology, it was still extremely helpful in our overall development of the project, and will follow us into industry in the future.
- GitHub has a limit to the size of file that can be uploaded, meaning that many of the ml-agents files could not be uploaded directly, as they exceeded the 100MB allowance

The whole development of the project was a challenging and rewarding experience. Although things didn't work out as we initially intended, and many aspects of the development cycle were rather challenging with a lack of useful documentation to work with, we all gained a great insight into the technologies used, as well as a greater bond supporting each other through the many struggles and hardships, whilst managing to keep a positive outlook throughout the development. Overall we are proud with the research and development we managed to fulfil, and we all have learned a lot despite the project not coming together as we had initially hoped.

# Appendices

**Github Repository Link** Below is a link for our github repository where the project is stored:

```
https://github.com/BrianD147/4th-Year-Project
```

**Installation Guide**
Download and install Anaconda for Windows using the following link:

```
https://www.anaconda.com/distribution/\#windows
```

Once Anaconda is installed you will need to run Anaconda Navigator to finish the setup.
After running Anaconda Navigator you can close it again. If environment variables were for some reason not created you will see error ”conda is not recognized as internal or external command” when you type “conda” into the command line. To fix this type “environment variables” into the windows search bar, and You should see an option called “Edit the system environment variables”. From here click “Environment Variables” and double click “Path” under “System variable. Click “New” and add the following new paths:

```
%UserProfile%\Anaconda3\Scripts
%UserProfile%\Anaconda3\Scripts\conda.exe
%UserProfile%\Anaconda3
%UserProfile%\Anaconda3\python.exe
```

Once this is all done, open the new “Anaconda Prompt” from the windows search bar. Then type in the following command:

```
conda create -n ml-agents python=3.6
```

If you are asked to install any new packages simple type “y” and press enter.
Now you’ll have to activate this new environment. In the same Anaconda Prompt type the following:

```
activate ml-agents
```

Finally lets install TensorFlow using the “pip” package manager in Anaconda. To do so type the following command from the same Anaconda Prompt:

```
pip install tensorflow==1.7.1
```

Now with all of that out of the way, we can move on to setting up the project files. Begin by cloning the GitHub repository, this can be done from the command line using the following command:

```
git clone https://github.com/BrianD147/4th-Year-Project
```

Once the repo has been cloned, you will need to retrieve the ML-Agents assets folder from the following Google Drive link:

```
https://drive.google.com/open?id=1mqoev-1K-7lDgqTIGWRbmnrZZCFxhKZY
```

This folder needs to be placed into the Assets folder of the project; this folder can be found here:

```
\4th-Year-Project\4th-Year-Project\{Assets\
```

Now with the files set up, you can begin by running a notebook and observing the environment. Navigate to the 4th-Year-Project folder in the command line, and navigate into the notebooks folder using the following command:

```
cd notebooks
```

Then type the following to run Jupyter:

```
jupyter notebook
```

This will open a web browser, from here select the 10-Episode-Random.ipynb file, this is a notebook which will connect to the Unity environment and send random inputs to the agent. Press the “Restart the kernel and run all cells” option from the top of the window (the button is two arrows pointing to the right).

When the configuration panel pops up, simple click “Play!”. (Note: if you are NOT using a graphics card it is recommended to lower the quality before running the environment)

If you would like to view the project working using the internal C# code, then simply open the Unity editor and run the “Soccer” scene. The C# code will control the agents and replicate what the neural network aimed to achieve.

# Bibliography

[1] "Unit Testing." `http://softwaretestingfundamentals.com/unit-testing/`. [Online; accessed 22-March-2019].

[2] "System Testing." `http://softwaretestingfundamentals.com/system-testing/`. [Online; accessed 22-March-2019].

[3] "Regression Testing." `http://softwaretestingfundamentals.com/regression-testing/`. [Online; accessed 22-March-2019].

[4] Unknown, "What is Regression Testing? Definition, Test Cases (Example)," [Online; accessed 18-January-2018].

[5] J. McCarthy, "Artificial intelligence, logic and formalizing common sense," in *Philosophical logic and artificial intelligence*, pp. 161–190, Springer, 1989.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[7] K. Hornik, C. Buchta, and A. Zeileis, "Open-source machine learning: R meets weka," *Computational Statistics*, vol. 24, no. 2, pp. 225–232, 2009.

[8] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems.* " O'Reilly Media, Inc.", 2017.

[9] "Unity Website." `https://unity3d.com/unity`. [Online; accessed 5-November-2018].

[10] J. Petty, "What is Unity 3D & What is it Used For?," [Online; accessed 5-November-2018].

[11] "What is Unreal Engine 4?." `https://www.unrealengine.com/en-US/what-is-unreal-engine-4`. [Online; accessed 8-November-2018].

[12] "Unreal Engine 4 - Overview." `https://hub.packtpub.com/overview-unreal-engine/`. [Online; accessed 8-November-2018].

[13] "What is Python? Executive Summary." `https://www.python.org/doc/essays/blurb/`. [Online; accessed 27-October-2018].

[14] "What is Python? Python For Beginners." `https://www.pythonforbeginners.com/learn-python/what-is-python/`. [Online; accessed 28-October-2018].

[15] "What is R?." `https://www.r-project.org/about.html`. [Online; accessed 29-October-2018].

[16] "What is R Programming Language? Introduction & Basics." `https://www.guru99.com/r-programming-introduction-basics.html`. [Online; accessed 29-October-2018].

[17] "A Brief Introduction to Lisp." `https://courses.cs.vt.edu/~cs1104/TowerOfBabel/LISP/Lisp.outline.html`. [Online; accessed 8-October-2018].

[18] "LISP Tutorial." https://www.tutorialspoint.com/lisp/. [Online; accessed 8-October-2018].

[19] "Jupyter Website." https://jupyter.org/. [Online; accessed 14-October-2018].

[20] "What is the Jupyter Notebook? - Documentation." https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#what-is-the-jupyter-notebook. [Online; accessed 15-October-2018].

[21] M. Driscoll, "Jupyter Notebook: An Introduction," [Online; accessed 14-October-2018].

[22] "TCP (Transmission Control Protocol) - Searching Network." https://searchnetworking.techtarget.com/definition/TCP. [Online; accessed 2-November-2018].

[23] "Transmission Control Protocol (TCP) - sdx central." https://www.sdxcentral.com/term/transmission-control-protocol-tcp/. [Online; accessed 2-November-2018].

[24] "Connecting Unity simulation with python controller over TCP - Stack Overflow." https://stackoverflow.com/questions/42531691/connecting-unity-simulation-with-python-controller-over-tcp. [Online; accessed 2-November-2018].

[25] "Iron Python." https://ironpython.net/. [Online; accessed 23-October-2019].

[26] M. Foord and C. Muirhead, *IronPython in action*. Manning Publications Co., 2009.

[27] "ML-Agents Github Repo." `https://github.com/Unity-Technologies/ml-agents`. [Online; accessed 5-December-2018] - Has since been updated.

[28] "Tensorflow." `https://www.tensorflow.org/`. [Online; accessed 22-November-2019].

[29] "Tensorflow - Tutorial." `https://www.tensorflow.org/tutorials`. [Online; accessed 22-November-2019].

[30] "Sony Neural Networks." `https://github.com/sony/nnabla`. [Online; accessed 2-October-2019].

[31] "Unity Test Tools." `https://unity3d.com/unity/qa/test-tools`. [Online; accessed 14-September-2019].

[32] "Docker." `https://www.docker.com/`. [Online; accessed 27-September-2019].

[33] "Docker - Getting Started." `https://www.docker.com/get-started`. [Online; accessed 27-September-2019].

[34] "Anaconda." `https://www.anaconda.com/`. [Online; accessed 18-November-2019].

[35] "Anaconda." `https://www.anaconda.com/why-anaconda/`. [Online; accessed 8-November-2019].

[36] J. R. Koza and J. P. Rice, "Genetic generation of both the weights and architecture for a neural network," in *IJCNN-91-seattle international joint conference on neural networks*, vol. 2, pp. 397–404, IEEE, 1991.

[37] D. F. Specht, "A general regression neural network," *IEEE transactions on neural networks*, vol. 2, no. 6, pp. 568–576, 1991.

[38] A. Krogh and J. Vedelsby, “Neural network ensembles, cross validation, and active learning,” in *Advances in neural information processing systems*, pp. 231–238, 1995.

[39] j. v. u. ”vincentpierre, dericp, “ML-Agents Toolkit - Academy.” `https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Learning-Environment-Design-Academy.md`. [Online; accessed 5-December-2018] - Has since been updated.

[40] a. m. d. u. x. ”Jo3w4rd, vincentpierre, “ML-Agents Toolkit - Academy.” `https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Learning-Environment-Design-Brains.md`. [Online; accessed 5-December-2018] - Has since been updated.