jamk.fi

Creating Self-Learning AI using Unity Machine Learning

Olli-Pekka Juhola

Bachelor's thesis
September 2019
Information Technology
Bachelor of Engineering

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Description

Author(s)	Type of publication	Date
Juhola, Olli-Pekka	Bachelor's thesis	September 2019
		Language of publication. English
	Number of pages 61	Permission for web publication. x

Title of publication

Creating Self-Learning AI using Unity Machine Learning

Degree programme

Information and communications technology

Supervisor(s)

Mika Rantonen, Jani Immonen

Assigned by

Abstract

The objective was to develop artificial intelligence for a simple game by using two distinct methods with the Unity Game Engine and comparing the results with each other.

The implementation methods used were Unity Game Engine's own Navigation and Pathfinding system and a toolkit made by Unity that allows the use of Machine Learning inside Unity projects. The main goal was to achieve working prototypes using both methods with the artificial intelligence seeking out and collecting coins in the environment and to demonstrate how both methods were implemented.

As a result, two functioning games were developed with the two different artificial intelligence methods and in both, the artificial intelligent agent will seek out and collect the coins as it should. The Machine Learning method used reinforced learning in the training of the artificial intelligent agents, where the agent did not know anything about the game except how to move after which it trained itself in the environment hundreds of times in a minute to figure out the core of the game. Unity Navigation and Pathfinding method was created using Unity's Navigation Mesh system, which allows the artificial intelligence to figure out where it can move as well as know the location of the coins immediately.

The two methods were placed against each other in a simple test to see how many coins both artificial intelligence methods would collect during a few minutes. The amount of coins collected by each method resulted in a fascinating score for the Machine Learnt artificial intelligence, which was able to beat Unity Engines own Navigation and Pathfinding system hands down. The use of Machine Learning in creating artificial intelligence in Unity was trickier to create; however, the results there were much more fascinating.

Keywords/tags (subjects)

Machine Learning, Artificial Intelligence, AI, Unity, Unity3D, Games, Programming

Miscellaneous (Confidential information)

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Juhola, Olli-Pekka	Opinnäytetyö, AMK	syyskuu 2019
		Julkaisun kieli Englanti
	Sivumäärä 61	Verkkojulkaisulupa myönnetty: x

Työn nimi
Creating Self-Learning AI using Unity Machine Learning

Tutkinto-ohjelma
Tieto- ja viestintätekniikka

Työn ohjaaja(t)
Mika Rantonen, Jani Immonen

Toimeksiantaja(t)

Tiivistelmä

Tarkoituksena oli kehittää tekoäly yksinkertaiselle pelille käyttämällä kahta erillistä menetelmää Unity-pelimoottorin kanssa ja verrata tuloksia toisiinsa.

Toteutustapoina käytettiin Unity-pelimoottorin omaa navigointi- ja polkujen hakujärjestelmää sekä Unityn valmistamaa työkalua, joka mahdollistaa koneoppimisen käytön Unity-projekteissa. Päätavoitteena oli saada aikaan toimivia prototyyppejä käyttämällä molempia menetelmiä tekoälyn etsiessä ja kerätessä kolikoita ympäristössä sekä pystyä osoittamaan, kuinka molemmat menetelmät toteutettiin.

Seurauksena oli, että kehitettiin kaksi toimivaa peliä erilaisilla tekoälyn menetelmillä, ja molemmat tekoälyagentit etsivät ja keräsivät kolikot toivotulla tavalla. Koneoppimismenetelmässä käytettiin vahvistettua oppimista tekoälyn koulutuksessa, jolloin agentti ei tiennyt pelistä mitään muuta kuin miten liikkua. Tämän jälkeen se harjoitteli itsenäisesti ympäristössä satoja kertoja minuutissa saadakseen selville pelin ytimen. Unityn navigointi- ja polunmääritysmenetelmä luotiin Unityn navigointiverkkojärjestelmällä, jonka avulla tekoäly ymmärsi, minne pystyi liikkumaan ja tiesi kolikoiden sijainnin automaattisesti.

Nämä kaksi menetelmää asetettiin toisiaan vastaan yksinkertaisessa testissä, jotta pystyttiin näkemään, kuinka monta kolikkoa molemmat tekoälyn menetelmät keräsivät muutaman minuutin aikana. Kullakin menetelmällä kerätyistä kolikoista koneoppinut tekoäly sai huomattavan määrän enemmän kolikoita ja pystyi lyömään Unity-moottorin oman navigointi- ja polkujärjestelmän. Koneoppimisen käyttö tekoälyn luomisessa Unity-pelimoottorissa oli vaikeampaa, mutta tulokset olivat paljon kiehtovampia.

Avainsanat (asiasanat)
Koneoppiminen, Tekoäly, Unity-pelimoottori, Unity

Muut tiedot

Contents

Figures

Glossary

Academy	A Unity ML-Agents component which controls the timing, resetting and training settings of an environment
Action	The agents' part of carrying-out of a decision in the learning environments
Agent	A Unity ML-Agents component, which produces the observations and actions in the environment.
AI	Artificial Intelligence
Brain	A Unity ML-Agents component, which makes the decisions for the agents it is linked to.
Environment	Unity scene which contains the Agents, Brains and Academy
FixedUpdate	A method in Unity, which is called each time the game engine is stepped. *FixedUpdate* -method is called before the *Update* -method
Game View	Interface window in the Unity Editor, which displays the rendered world through the camera, best described as "What the user sees"
GameObject	Base asset in Unity that contains different components from physics to scripts
Hierarchy Window	Interface window in the Unity Editor, which contains every object in the scene
Inspector Window	Interface window in the Unity Editor, which displays the components of a GameObject
ML	Machine Learning

Policy	Function for producing the decisions of agents from observations
Prefab	A prefabricated GameObject, which can be easily saved, edited and duplicated in order to save the developer's time
Project Window	Interface window in the Unity Editor, which displays one's library of assets.
RigidBody	A component in Unity, which handles physics
RTS	Real-Time-Strategy
Scene View	Interface window in the Unity Editor, which allows to navigate and edit the scene visually.
Scene	A place to build your environment in the Unity Engine
SDK	Source Development Kit
Std of Reward	Standard deviation of the reward, a measure of the spread around the mean reward.
Step	Corresponds to each *FixedUpdate* call of the Unity Game Engine.
Unity	A popular game engine created by Unity Technologies
Update	A method in Unity, which corresponds to a frame

1 Introduction

When talking about Artificial Intelligence (AI), most people think of video games. The roots of video game AI can be traced back to 1950s when Claude Shannon and Alan Turing began to write AI logic for chess programs. In 1997, the famous computer "Deep Blue" that represented the pinnacle of AI techniques beat the chess Master Garry Kasparov in a publicized match. Developing AI for turn-based tabletop games back in the day was much simpler than developing AI for real-time strategy (RTS) games, since the first RTS games, for example Warcraft (1994), did not have impressive AI logic and were said to be more like "puzzles" than "war games". (Middleton 2002)

Pattern recognition is a basic human skill for humans; however, it can be challenging for computers. Sequential decision-making is a core topic in machine learning. The ability to let the AI decide on its own is a fascinating concept, and it is progressively being worked on in every field including video games. Creating simple traditional AI in videogames can be slightly tricky sometimes. Most of the times AI will be a mindless zombie always going to the target location and using the fastest path to the target. (Middleton 2002)

This paper discusses the use of AI and machine learning in Unity game engine. Using Unity's own machine learning toolkit, the primary plan is to research the basics of Unity machine learning, which makes it possible to train intelligent agents using reinforcement learning (RL) via simple Python API and use it to implement a self-learning AI into a Unity game.

This paper seeks answers to the following research questions.

- What is Machine Learning?
- How can Machine Learning improve AI in games made with Unity?
- How will Machine Learning Agents behave compared to traditional Unity AI?

To answer these questions, there will be two simple test environments made using the Unity Machine Learning Agents (Unity ML-Agents) toolkit as well as the traditional Unity Navigation system. In the theory section, the structure and basic func-

tionalities of the Unity Engine are covered, followed by familiarizing the main principles of Machine Learning and its basic functionalities as well as getting a brief overview of the Unity ML-Agents toolkit. The Unity Engine and Unity ML-Agents toolkit basis of knowledge were accumulated by researching and analyzing the online documentation that was created by Unity Technologies. Machine Learning basis of knowledge was accumulated by researching online articles about the workflow. The second part of the paper covers the installation process and implementation of the Unity ML-Agents toolkit as well as building an example environment for the AI agents in which to learn.

2 Unity Development Platform

2.1 Introduction

” Start *bringing your vision to life today. Unity’s real-time 3D development platform empowers you with all you need to create, operate, and monetize.”* (Unity 2019).

The Unity development platform is a powerful all-in-one editor that empowers developers with everything one needs to create, operate and monetize one’s products. Unity was originally released back in 2005 to help develop a game called Goo Ball; now it is one of the world’s leading real-time creation platforms. With the Unity Editor, one can achieve a plethora of possible products ranging from games all the way to animated films and engineering. The powerful all-in-one editor includes a vast range of artist-friendly tools implementation of high-performant gameplay logic. (Unity 2019.)

Unity is a free product that can be used by everyone with vast documentation of every single expression, mechanic or system the engine provides, with more additions constantly being developed and released. Unity’s free license is valid only if one’s revenue or funding does not exceed over $100,000 annual gross revenue, after which one is obliged to purchase either the Plus-edition of Unity (which covers up to $200,000) or Pro-edition with no revenue cap. (Unity License Agreement 2019.)

2.2 Terminology

When creating a Unity Project, it consists of a plethora of different assets. An asset corresponds to a file inside the project, which can be anything from a 3D model to an audio clip. Most common assets in a project are GameObjects. A GameObject is an object inside the project that can consist of different components. Unity Editor offers a vast number of different components to use in projects that can be attached to one’s GameObjects including Cameras, Renderers, RigidBodies, Colliders and more. Furthermore, it is also possible to define and create custom components using C# scripts, which allows the creation of plugins and editor extensions.

2.3 Unity Editor

2.3.1 Main Features

The powerful all-in-one Unity Editor allows to create anything one can imagine, from games to animated films and from Virtual Reality (VR) to Augmented Reality (AR). Figure 1 presents a screenshot of the Unity Editor. With Unity's efficient workflows comes Unity's Prefab system. Prefabs are GameObjects that have been preconfigured to work the way the creator wants them to, allowing creators to use said Game Objects more easily and efficiently. (Unity 2019.)

The Unity Editor is also equipped with powerful physics engines, a built-in User Interface (UI) system to create intuitive UI and custom tools to expand the editor itself visually and create powerful new tools to help your team achieve tasks faster. (Unity 2019.)

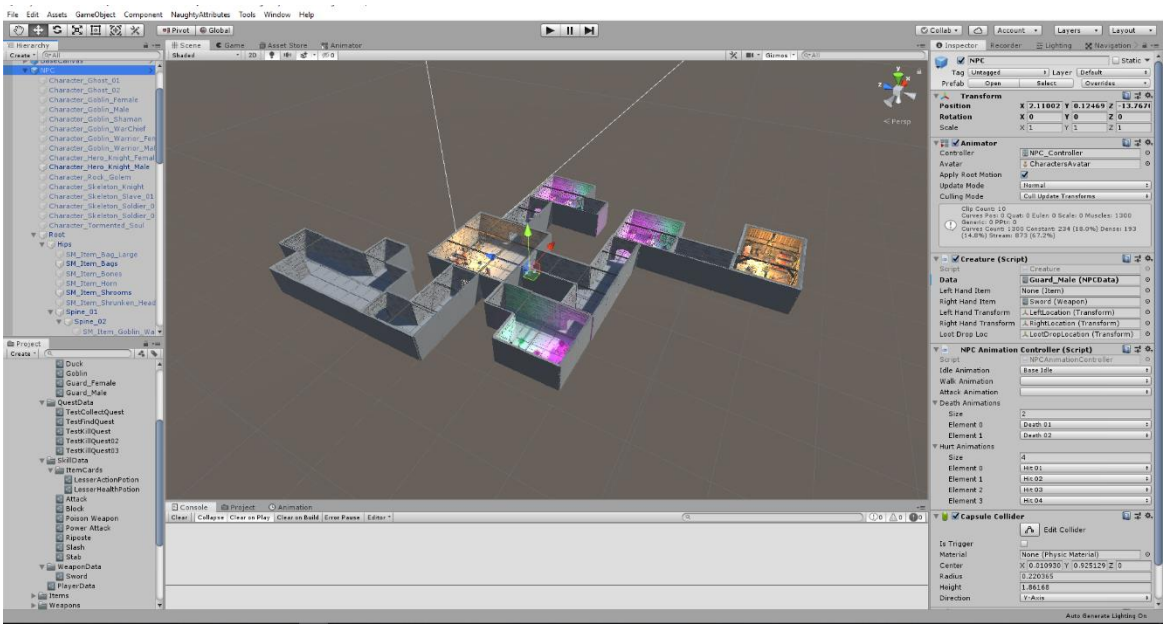


Figure 1. The Unity Editor

2.3.2 Multiplatform

Unity is an industry-leading multiplatform that allows the creators to deploy to a range of over twenty-five different platforms, including mobile, desktop, console, TV, VR and much more. Unity has more platform support than any other creation engine

and allows the creators to deploy to any platform using a single build of your product. (Unity 2019.)

2.4 Unity as a Game Engine

2.4.1 Introduction

Unity has everything creators need to succeed in game development. Unity Editor can be extended to efficiently improve a developer's or a team's workflow, bringing new content or mechanics into the games much more reliably. Unity as a game engine has grown into one of the world's most popular game engines, and Unity Technologies CEO John Riccitiello stated that almost half of all games in the world are made with Unity, mainly because Unity games can be easily published on multiple platforms, something that other game engines still have problems with. (Dillet 2018.)

2.4.2 Scripting

GameObjects in Unity have a vast number of components to choose from; however, sometimes more is needed than what Unity offers. In Unity, one can create custom scripts, which allows the creation of new components that sometimes must be specific for a specific GameObject. The main programming language for Unity is C#, which is an industry-standard language. (Creating and Using Scripts 2019.)

Unity comes with its own integrated scripting editor called MonoDevelop. It is possible for users to use their preferred editor as well.

Unity Technologies is currently developing a Visual Scripting tool to use in the Unity Editor. This will allow most users with little to no programming experience to create scripts using a visual node-editor. Before this, users would have had to either create their own Visual Scripting tools or purchase a Visual Scripting editor tool from the Unity Asset Store.

2.4.3 Scenes

In Unity, scenes are special assets that are the main element of implementing anything to a game. When a scene is first created, there will automatically be a camera

and a light source inside the scene, the camera will render everything the users see in their game as was illustrated in Figure 1. Inside the scene, one can start adding GameObjects from 3D models to UI. (Scenes 2019.)

The best way to summarize a scene is to think of it as the area users play with their character or design their world. The best way to optimize a game is to have multiple scenes in the project, for example if there is a scene with a large outside environment and one wants to transfer one's player character indoors, thus one could create a new scene which only contains the indoor assets. Building and adding everything in once scene will be very performant heavy and require plenty of processing power.

2.5 NavMeshAgent

Unity Engine comes with its own workflow for creating AI called NavMeshAgent. A NavMeshAgent is a component attached to a character in the game, which allows said character to navigate the environment using Navigation Mesh (NavMesh). NavMesh is an abstract data structure used to aid agents in pathfinding through environments. NavMesh is a collection of two-dimensional (2D) polygons that inform the agents which areas of the environment can be traversed as illustrated in Figure 2. In Unity, the NavMesh can be automatically baked into the environment from the environments level geometry. (Navigation and Pathfinding 2019.)

NavMesh baking can also detect obstacles in the environment, which are marked with a NavMeshObstacle component to recognize said object as an obstacle when baking the NavMesh onto the environment. The Off-Mesh Link component as shown in Figure 2 allows the incorporation of shortcuts onto the NavMesh which otherwise cannot be represented using a regular walkable surface. The Off-Mesh Link component best uses include climbable/jumpable areas as well as opening a door before walking through it. (Navigation and Pathfinding 2019.)

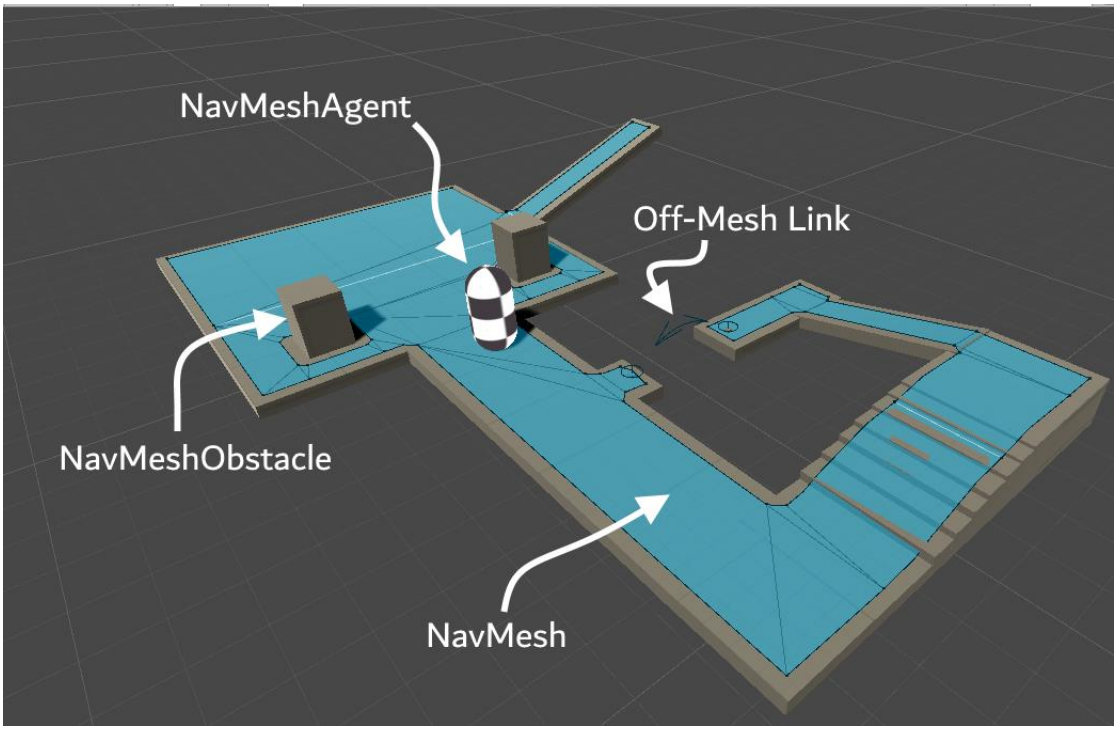


Figure 2. Unity Navigation System

3 Machine Learning

3.1 Introduction

Machine learning is a field of study, which gives computers the ability to learn without being explicitly programmed. It is essentially finding meanings in large amounts of data after which the machine starts making forecasts and predictions based on the data. Machine learning can be referred to as statistical modelling. Statistical Modeling is used to verify collected data, correct or delete any incomplete data and use the collected data to test hypotheses and make predictions. With machine learning this statistical modelling procedure is flipped making it so that the data determines which technique should be selected to get the best possible outcome of the task. (Esposito, Bheemaiah & Tse 2017.)

Arthur Lee Samuel was an award-winning pioneer in the field of AI and computer gaming. Samuel was the one who invented the term “machine learning” in 1959. Samuel was among the first who created a successful self-learning program for playing computer checkers. (Wiederhold, McCarthy & Feigenbaum n.d.)

There are three main types of machine learning. supervised machine learning based on task driven labeled datasets, unsupervised unlabeled datasets that cluster data together using identification algorithms and reinforcement learning where an artificial intelligent agent will be rewarded for achieving a given task. This is illustrated in Figure 3. (Dwivedi 2018.)

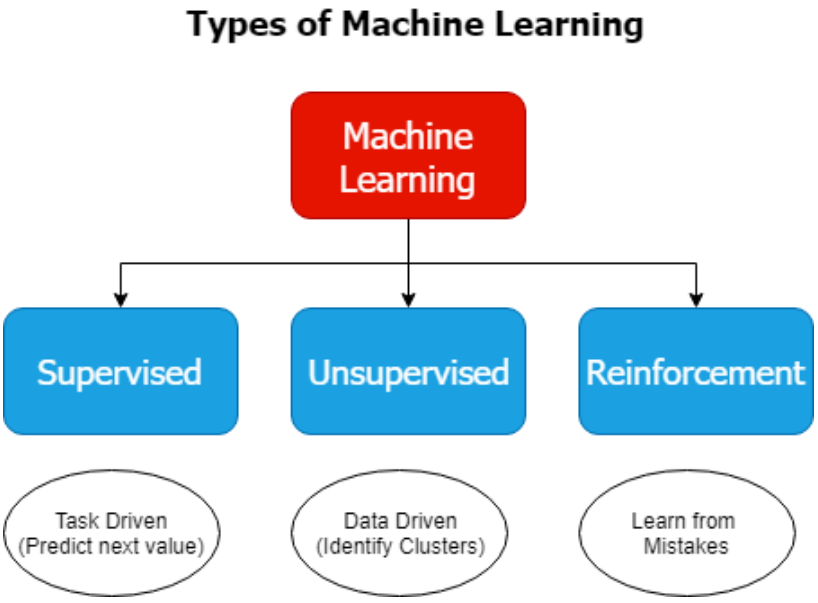


Figure 3. Types of Machine Learning

3.2 Supervised Learning

Supervised learning in machine learning labels the data given to tell the machine exactly what to search for. It requires a full set of labeled data while training an algorithm. The algorithm observes and identifies specific patterns in the given data allowing the computer to learn from the observations. With this, the computer improves the prediction performance the more observations it goes through. Figure 4 shows an example of supervised learning. (Salian 2018.)

An example of supervised machine learning could be to give it a labeled dataset of photographs of cars and tell the model which of the photographs were Audis, Volvos or Toyotas. When given another photograph, the model will then compare it to the previous dataset given in the training phase to predict the correct label of the car. (Salian 2018.)

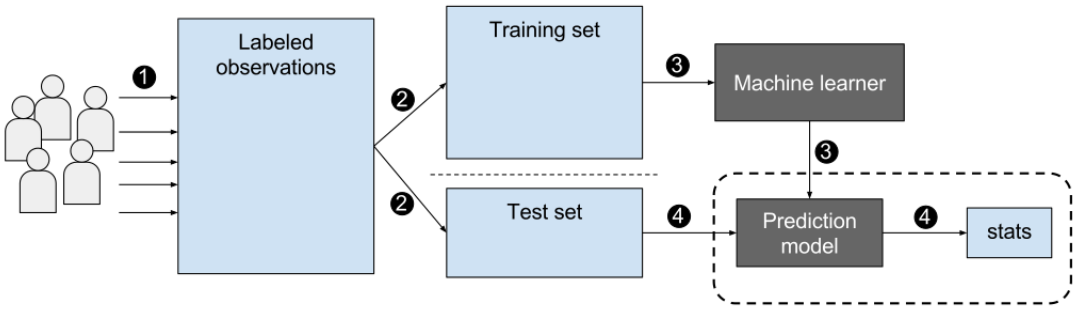


Figure 4. Supervised Learning Example

3.3 Unsupervised Learning

Supervised learning was given a full labeled dataset to examine and identify the same patterns in given data, whereas the unsupervised learning goes the other way around. Unsupervised learning is handed an unlabeled dataset with no instructions on what the machine is supposed to do with it. The training dataset does not contain any specific outcome or correct answer unlike supervised learning and with the dataset the neural network will automatically try to find structure in the given data. (Salian 2018.)

Unsupervised learning will automatically find and extract specific features and patterns that is sees in the given data. Clustering is a way for unsupervised learning model to organize the data relying on specific features that it then groups up together. An example for this could be to have a dataset of different animals, where the algorithm then tries to match every species to its own species group as illustrated in Figure 5. (Salian 2018.)

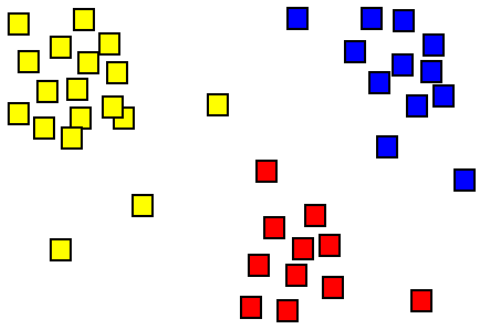


Figure 5. Clustering Data

3.4 Reinforcement learning

Reinforcement learning (RL) is where an agent makes a sequence of decisions to achieve a given goal in an environment. The AI agent will go through trial and error to come up with the best possible solution to a problem and because the agent is to be rewarded for the right actions, it performs and gets penalties for negative actions it should not do. This way the AI agent will do what the programmer wants. Currently, reinforced learning is a powerful way of letting a machine control an agent in a creative way. Using reinforced learning, an AI agent can undergo and experience years' worth of training in a short time span. (Budek & Osiński 2018.)

In 2018 OpenAI, an independent research institute to advance AI, created AI bots that played and won against a semipro team in Dota 2. Dota 2 is a popular videogame made by Valve, where two teams of five play against each other in a competitive match. The team the AI played against are in the top one percent (1%) on the Dota 2 global leaderboards. The AI bots were trained using reinforcement learning and learned to play Dota 2 by playing against itself for millions of times. Each day the AI would have played an equivalent of 180 years of Dota 2. (Simonite 2018.)

Reinforced learning has three main components. The agent, which is the decision maker in the learning model, the environment where the agent will interact with and actions which represents the interactions and everything the agent can do as seen in Figure 6. (Dwivedi 2018.)

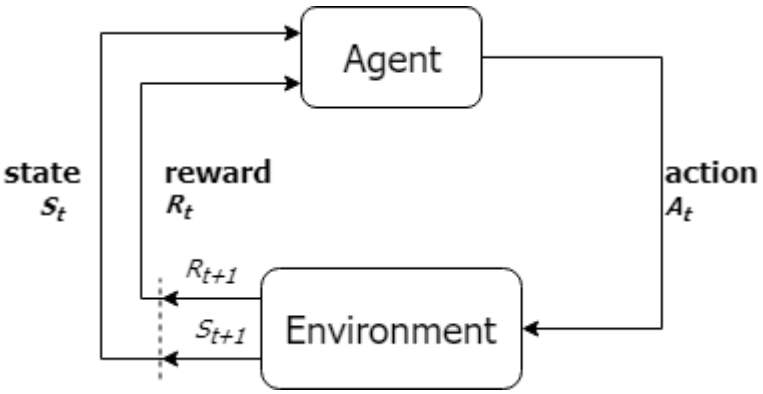


Figure 6. Reinforcement Learning

3.5 Unity ML-Agents Toolkit

3.5.1 Introduction

The Unity Machine Learning Agents Software Development Kit (ML-Agents SDK) is an open source plugin for Unity that enables the creation of simulation environments using the Unity Editor. Researchers and developers can easily interact with the ML-Agents simulation through a simple-to-use Python API. These agents can be trained using the reinforcement learning, imitation learning or other machine learning methods. The Unity ML-Agents toolkit offers a set of core functionalities, which enables creators to define an environment using the Unity Editor and associated C# scripts which can then expose the environments for interaction using the Python API. (Juliani, Berges, Vckay, Gao, Henry, Mattar & Lange 2018.)

Unity ML-Agents is based off TensorFlow, which is a machine-learning framework made by Google. TensorFlow allows developers to focus on the overall logic of their applications and not having to worry about implementing complex algorithms. (Yegulalp 2019.)

3.5.2 ML-Agents SDK

The ML-Agents SDK consists of three entities. Agent, brain and academy. Once the SKD package has been imported into a Unity project, creators can then modify scenes into a learning environment for the ML-Agents as seen in Figure 7. (Juliani et al. 2018.)

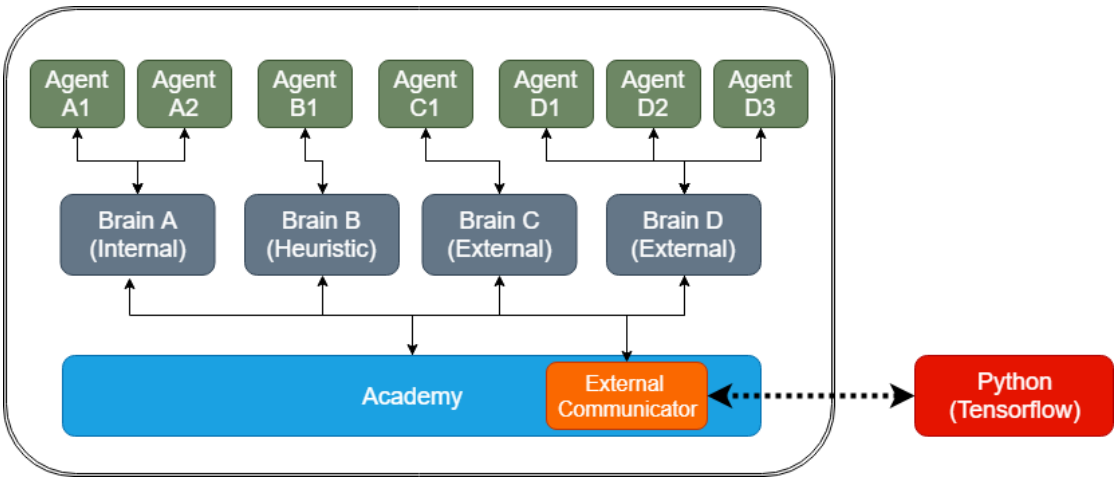


Figure 7. A Learning Environment containing agents, brains, and the academy

The first entity of the ML-Agents SDK is the agent component. The agent is simply just a GameObject in the scene; it is used to know what one's agents are in the learning environment. The agent GameObject is used for collecting necessary observations and executing actions within the learning environment. (Juliani et al. 2018.)

The second entity is the brain component. The brain component is used to make all decisions on the Agents. Without the brain, the agents are mindless. Each agent component is linked to a single brain that handles its decisions. There can be multiple agent components with the same brain, but an agent cannot have multiple brains. The decision making of a brain is providing a policy to the agents referred to as a Player, Heuristic, Internal, and External, which respectively means if they are conducted through player input, predefined scripts, internally embedded neural networks or interaction through the Python API. (ibid.)

The third entity is the academy components used within a scene to keep track of the steps of the simulation process. It can provide basic functionalities, for example the ability to reset a learning environment or modify the simulation speed. (ibid.)

3.5.3 TensorFlow

Unity ML-Agents is built on top of the open-source library TensorFlow, which is a library used for performing computations using data flow graphs. TensorFlow allows developers to focus on the overall logic of their applications and they do not have to worry about implementing complex algorithms.

When training with TensorFlow with the Unity ML-Agents toolkit, it outputs the trained model into a file (.nn) which can be used to attach to the brain Component.

TensorFlow comes with a nifty tool called TensorBoard. The data and statistics are saved each learning sessions that can be viewed easily with TensorBoard. With TensorBoard one can determine if the agent one trained had issues in the learning session, how long it took for the agent to learn and how well the agent was rewarded This is illustrated in Figure 8. (Unity ML-Agents GitHub n.d.)

Meaning for each graph as illustrated in Figure 8:

- *Environment/Cumulative Reward* – Each episodes' mean cumulative reward over all agents.
- *Environment/Episode Length* – Each episodes' mean length in the environment.
- *Environment/Lesson* – Plots the progress for each lesson. Used with Curriculum Training
- *Losses/Policy Loss* – How much the process for deciding actions is changing. Decreases during a successful training session.
- *Losses/Value Loss* -
- *Policy/Entropy* – How 'random' the decision of the agents is. Decreases slowly during a successful training session.
- *Policy/Learning Rate* – Determines how big a step the algorithm takes when searching for the optimal policy. Decreases over time.
- *Policy/Value Estimate* – The estimated mean value for all states visited. Increases during a successful training session.

Figure 8. TensorBoard example charts

4 Implementation

4.1 Installing the Unity Project Files

Installing the ML-Agents toolkit is very simple. Unity has created a repository on GitHub with the Unity project that contains the ML-Agents toolkit as well as the full documentation on how to install the mandatory Python packages. To start the process, it is obligatory to download the Unity ML-Agents repository from Unity's GitHub as demonstrated in Figure 9.

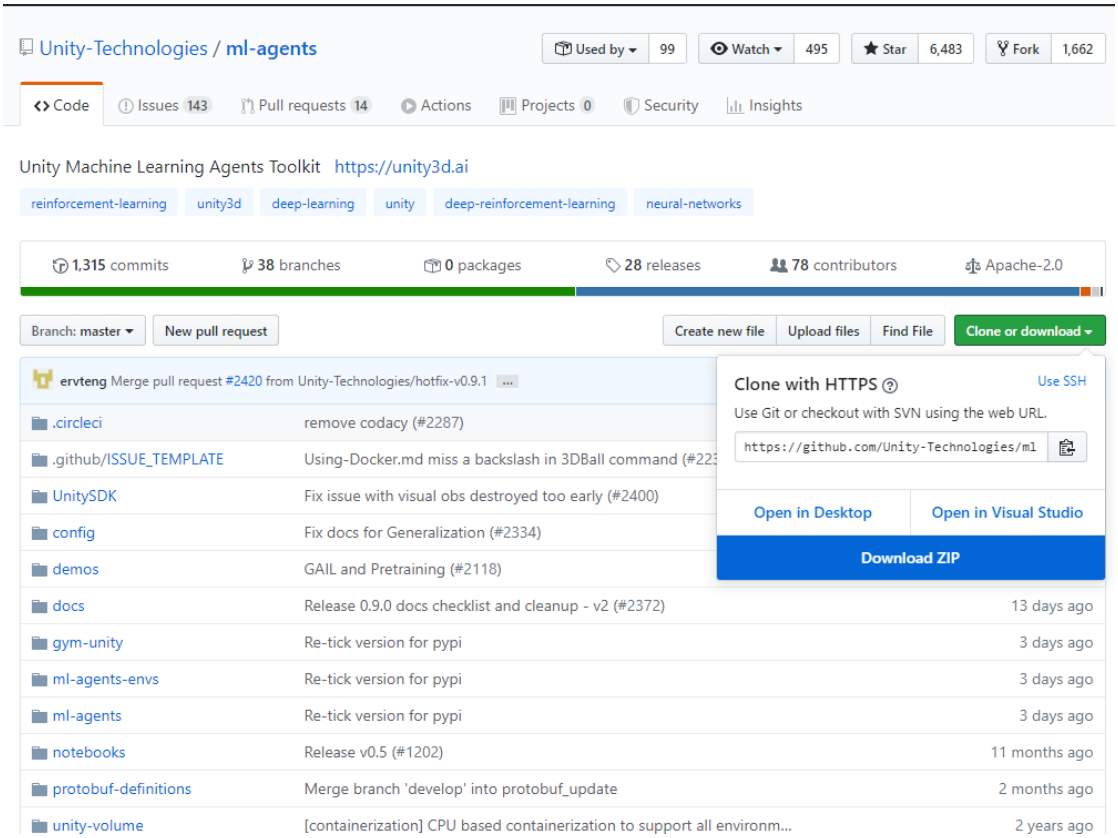


Figure 9. Unity ML-Toolkit Repository

Once the repository has been downloaded and unzipped, the Unity Project folder inside the repository will be linked to the Unity Hub, which contains everything a Unity Project requires as well as the Unity ML-Agents toolkit as seen in Figure 10.

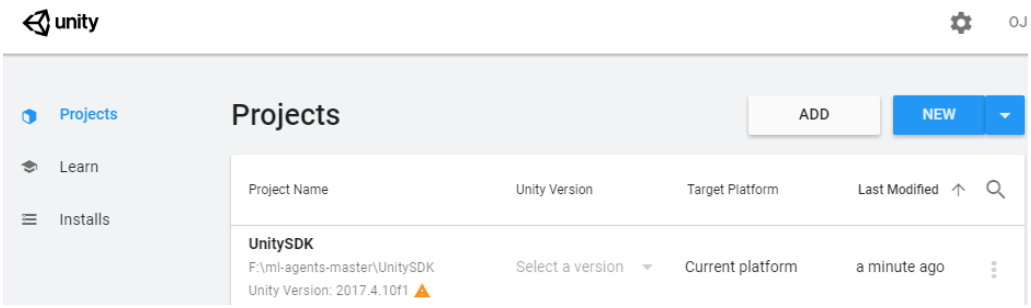


Figure 10. Project folder added to Unity Hub from the downloaded repository

Now that the project has been added to Unity Hub, it can be opened. Inside the projects Assets folder there is everything needed to start creating and experimenting with the learning environments inside Unity. It is now possible to test whether the Unity Project is working as intended, before the essential tools are installed.

Unity has created sample learning environments that contain Agents with pre-trained Brains. The examples vary in logic so the users can have an early look at what one can create with the Unity ML-Toolkit. The SoccerTwos environment is a good example to look at. The environment contains two teams competing each other in a game of soccer; each team has a goalie and a striker. The SoccerTwos scene is in the "ML-Agents/Examples/Soccer/Scene" path in the Project view of the Unity Editor as seen in Figure 11.

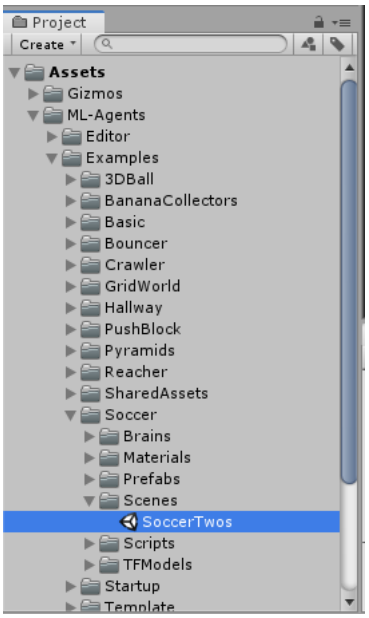


Figure 11. Locating the Soccer example scene in the Project View.

After opening the soccer learning environment, the scene will be loaded in and the “Scene View” now shows the learning environment as seen in Figure 12. First time when loading the environment, the Agents in the scene are portrayed as the blue and red rectangles. The goalie and striker agents both have different brain components, but both teams’ goalies share the same brain component. For the brains to function they need be assigned a training model, which are located in the “TFModels” folder under the “Soccer” folder as seen in Figure 13 and move the models into their corresponding brains in Figure 14.

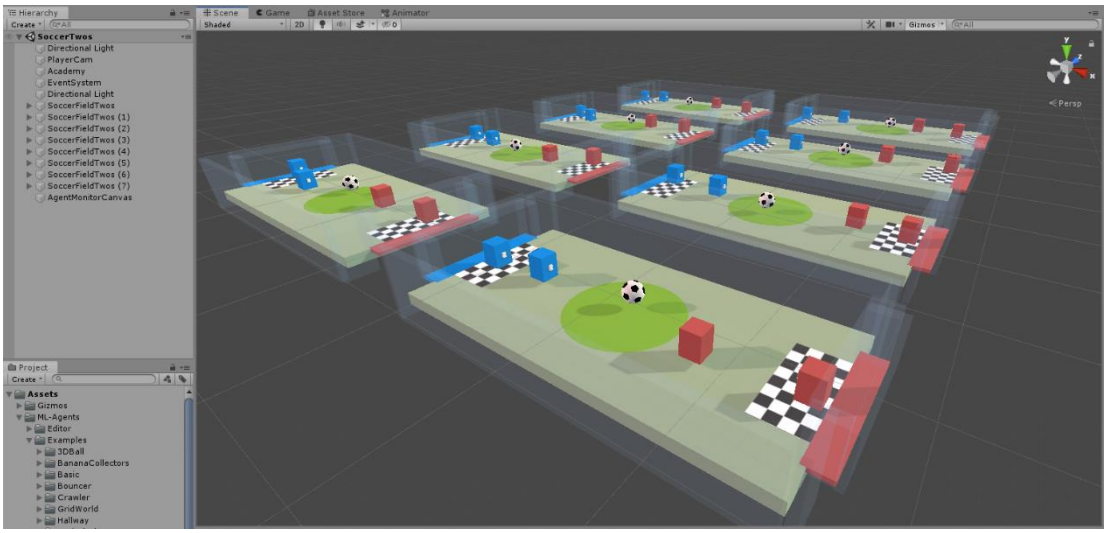


Figure 12. Scene View after opening the soccer learning environment scene

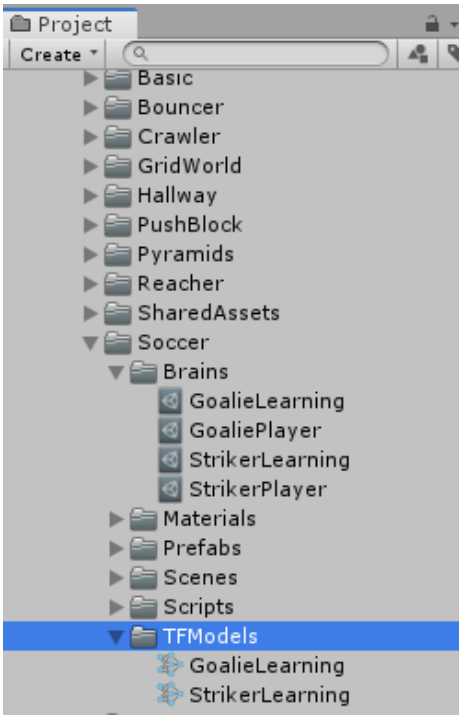


Figure 13. TFModels folder in the project view

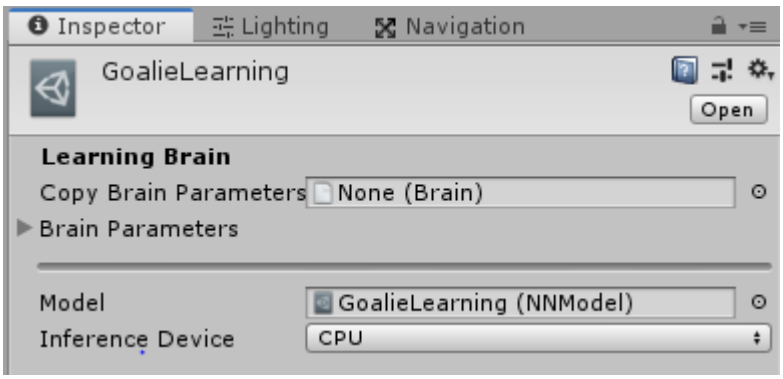


Figure 14. Moving the "GoalieLearning" Model into the GoalieLearning brain Component

After moving both the GoalieLearning model as well as the StrikerLearning model into their corresponding brains, hitting the “Play” button in the Unity Editor will launch the game and the two teams competing in soccer are immediately seen as demonstrated in Figure 15.

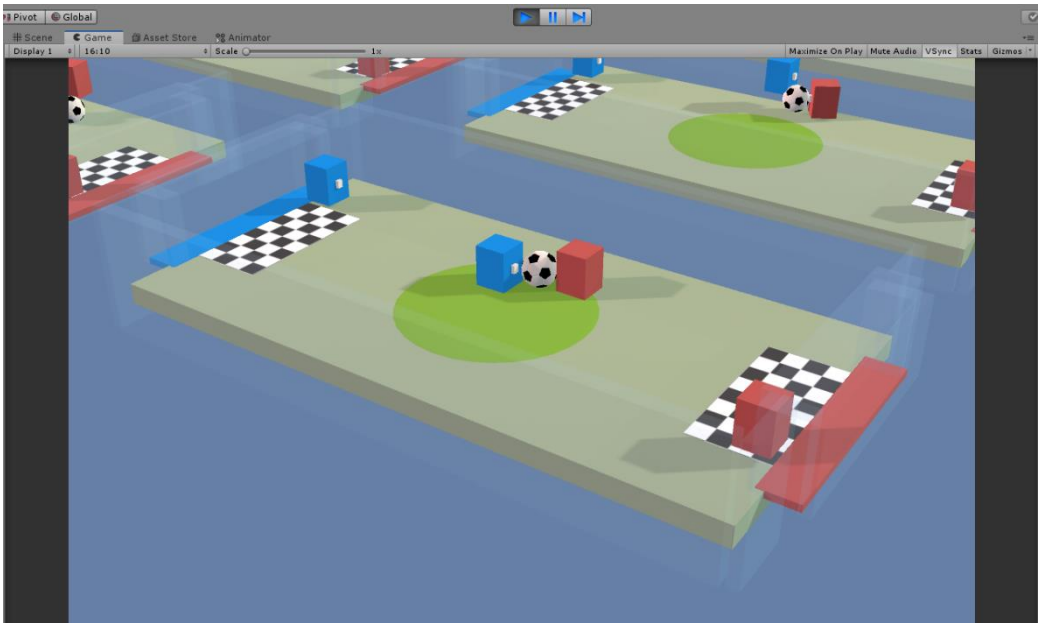


Figure 15. The Game View showing the two teams compete in soccer

4.2 Setup of the ML-Agents Toolkit

4.2.1 Installing Python via Anaconda

To start the process, it is essential to download and install Anaconda, an open-source distribution of Python for different scientific computing methods including machine learning. With Anaconda, it is possible to manage separate environments for different distributions of Python. Once Anaconda has been installed, it is a good idea to check if the installation added the mandatory system environment variables, otherwise it will not function properly. A relatively quick way to check if it has installed correctly is to type “conda” into the command prompt and if the command prompt displays this error, “conda is not recognized as an internal or external command, operable program or batch file.” It is then mandatory to add the system variables manually. The appropriate variables to add to the “Path” system variable are shown in Figure 16. Once the system variables are added, typing “conda” into the command prompt again should display the correct information.

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

Figure 16. Appropriate variables to add to the System Environment Variables

4.2.2 Setting up a New Conda Environment

To use the ML-Agents toolkit, it is vital to create a new Conda environment. A Conda environment means that all packages that are installed are localized to that environment and will not affect any other environments. The command to create a new Conda environment that used Python version 3.6 is. "*conda create -n ml-agents python=3.6"*. (Appendix 1 shows the full console view.)

To use the new Conda environment named ml-agents it needs to be activated by using the command line shown in Figure 17, which will prepend the "(ml-agents)" tag at the beginning of the second line. After activating the environment TensorFlow, the main framework needed for the ML-Agents must be installed. TensorFlow will be installed using the "pip" command, a package management system in Python used to installing different packages. According to the install documentations of Unity ML-Agents, the latest versions of TensorFlow are not supported, thus it is mandatory to install version 1.7.1 of TensorFlow for it to work. Using this command on the command prompt "*pip install tensorflow==1.7.1*" installs TensorFlow version 1.7.1. After TensorFlow has been installed, the rest of the essential Python packaged needed to run ML-Agents can now be installed by typing "*pip install mlagents"* in the Anaconda Prompt.

```
(base) C:\Users\Agamashi>activate ml-agents

(ml-agents) C:\Users\Agamashi>
```

Figure 17. Activating the environment. When activated the (ml-agents) will be prepended on the first line

4.3 Training the Brain with Reinforcement Learning

4.3.1 Setting up the environment for training

Before starting the creation of one's own environment, it is a good idea to check if everything is working as intended. The fastest environment to learn is the 3DBall environment, where the idea is for the Agents to balance a ball on top of a platform. Opening the 3DBall scene inside the Unity Project window under the "ml-agents/examples" folder opens the environment. The key requirements to start the learning process are to find the academy component in one of the GameObjects in the Hierarchy window. The GameObject containing the academy component is usually named "[Game]Academy", where the "[Game]" is the name of the project or game, for the 3DBall environment, the academy is called "Ball3DAcademy". When the GameObject containing the academy is selected, the inspector window now displays the academy script and under the "Broadcast Hub" section of the academy the "3DBallBrain" is added into the "Brains" property if it is not already there. The last step is to tick the box "Control" next to the "Brain" property, which means the brain that is set in the property will be the one that is going to get trained as seen in Figure 18. (A full screenshot of the editor is seen in Appendix 2)

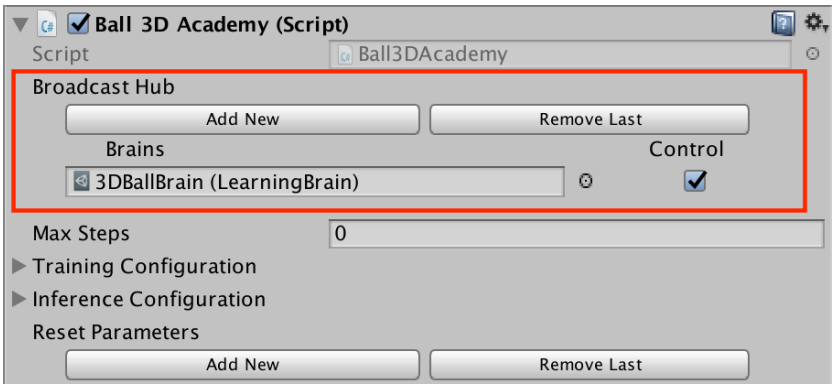


Figure 18. The Ball3DAcademy script component in the Inspector view

4.3.2 Training the environment

After setting up the environment inside Unity, it is now possible to start the learning process using the Anaconda Prompt. If the Conda environment is not activated, it is

necessary to be active in order to use the commands necessary to start the training process. After the environment is up, running the command "*mlagents-learn <trainer-config-path> --run-id=<run-identifier> --train"*, where *"<trainer-config-path>"* is the relative file path of the trainer configurations, "*<run-identifier>"* is a string used to separate the results from different training sessions and *"—train"* tells the "*mlagents-learn"* to run the training session. Command used is shown in the snippet below.

```
mlagents-learn F:\ml-agents-master\config\trainer_config.yaml --run-id=firstRun --train
```

After typing the command inside the Anaconda Prompt, the console will now show information about the training that will take place. At the bottom of the console it says that pressing the "Play" button in the Unity Editor will start the training process (Appendix 3 shows the full console view.). After pressing the "Play" button in the Unity Editor, the game will launch, and the Anaconda Prompt will now display information on the current launched environment and display the test results of each training run (Appendix 4). From the Anaconda Prompt it is now possible to get the data from the training run and use TensorBoard to get the data into charts. (Appendix 5).

When the training process is finished, the training model is located inside the "ml-agents-master/models" folder, which will need to be moved inside the Unity Project. Once moved, it is now possible to use this model in the 3DBallLearning brain using the same method that was used in the SoccerTwos example, which is moving the model into the model property of the brain and pressing the play button. After training the "Control" property inside the "3DBallAcademy" GameObject must be unticked now in order to test the model inside the game. Pressing the "Play" button inside the Unity Editor now will launch the game and see that the platforms are balancing the balls on their own with the trained model that was added to their Brains as seen in Figure 19.

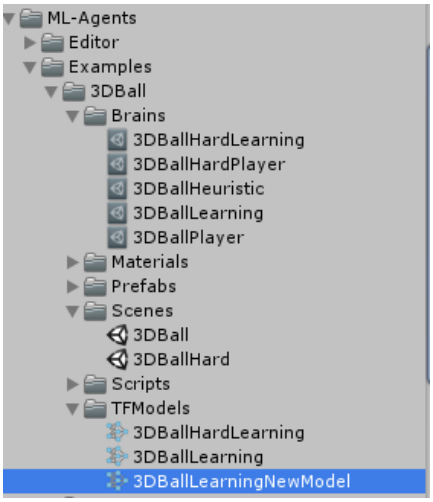


Figure 19. The new trained model for the 3DBallLearning brain

4.4 Creating and Designing a new Learning Environment

4.4.1 Adding the basic components

Creating a new learning environment for Unity ML-Agents only requires a few basic Unity elements as well as the agent, brain and academy components from the Unity ML-Agents toolkit. Unity ML-Agents comes with a template environment, which only contains the required Unity ML-Agent components as well as the scene. From here it is wise to duplicate the “Template” folder inside the “ML-Agents/Examples” folder and give it a new name, for this example it will be renamed “SimpleCollector” as demonstrated in Figure 20.

Figure 20. Folder containing the template assets

The template folder contains a template academy component that is wise to rename to use the same name as the current project, for this it will be renamed as “SimpleCollectorAcademy”. The academy script contains two methods, “AcademyReset” and “AcademyStep”; however, these methods will not be used in this example, so it is wise to remove them from the script as seen in Figure 21. This new academy script is now a child of the base “Academy” parent script containing the logic for the whole

academy system. This method is called inheritance, which allows defining a child class that inherits, extends and modifies the behavior of the parent class; however, for this example it is not necessary.

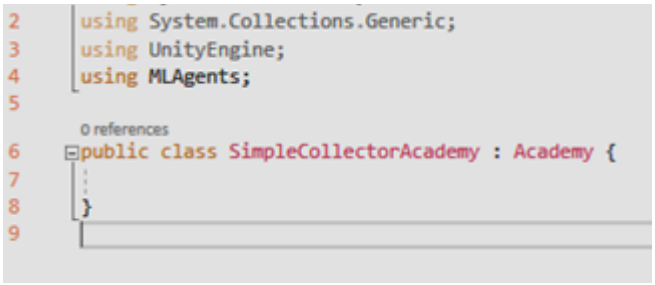


Figure 21. The SimpleCollectorAcademy script. Inherited from the base academy

This same procedure goes for the “TemplateAgent” script, which will be renamed as “SimpleCollectorAgent”. The script contains four methods. These methods are “CollectObservations” which handles the observations of the agent, “AgentAction” which handles the actions of the agents, “AgentReset” which handles the resetting of the agents and “AgentOnDone” which handles the finishing logic of the agents. For this example, the “AgentOnDone” method will not be used as shown in Figure 22.

Figure 22. The SimpleCollectorAgent script, inherited from the base Agent script

The third template script inside the template folder is the “TemplateDecision” script. This script is the base for manual decision-making and for the use of the heuristic brain. In this example, heuristic brain will not be used so it can be removed from the folder.

Before the logic for the agents can be made, the agents will need a brain. For this example, there will be two separate brains, one brain for learning and the other for the player. It is often wise to create a “Player” brain as well, since it is the easiest way to test if one’s logic for the agent works the way one wants it to. Creating the new brains is rather simple, the first thing to do is to create a new folder called “Brains” that contains the two brains as illustrated in Figure 23. The brains will be named “SimpleCollectorLearning” and “SimpleCollectorPlayer”. The result should look something like shown in Figure 24.

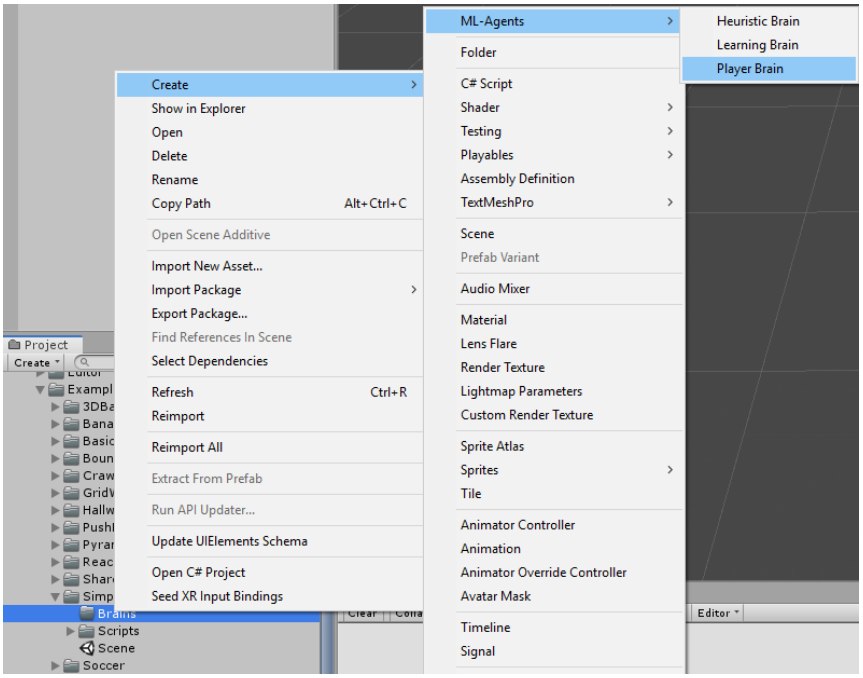


Figure 23. Locating and adding the Player brain as well as the Learning brain inside the Brains folder

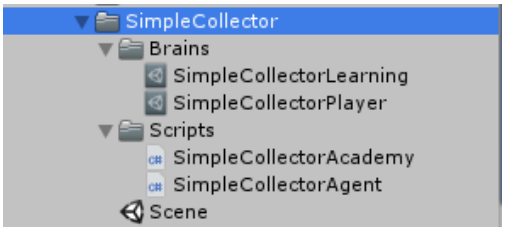


Figure 24. Result of the SimpleCollector folder after adding the brains

4.4.2 Creating the environment

The “SimpleCollector” environment will only contain a few GameObjects. The key components needed are the academy and agent components. For the academy component a simple empty GameObject can be created and after that, it is renamed “Academy”. After the academy GameObject has been created, it needs the corresponding academy component that was created. Simply dragging and dropping the “SimpleCollectorAcademy” script into the inspector view while the “Academy” GameObject is selected will work. After adding the component, the academy requires a brain inside the “Broadcast Hub” property of the script. At first, it will have the player brain as seen in Figure 25.

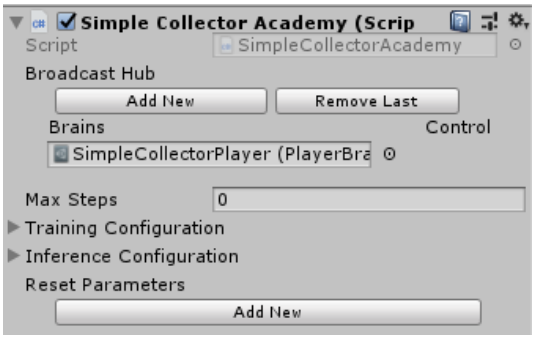


Figure 25. SimpleCollectorAcademy script after adding the player brain into the brain slot

After setting up the “Academy” Gameobject, it is time to create the agent. For this example, the agent will be a capsule. After creating a new capsule, it will be given a new name “CollectorAgent” as well as adding the corresponding agent script onto the GameObject. The “CollectorAgent” will also need physics in-order to move, so the Unity standard Rigidbody component will be added to the agent. The agent will also be given a new material in order to distinguish it from the game world. After adding the necessary components, the “Brain” property of the “SimpleCollectorAgent” script will need a brain in order to function; at first, it will have the player brain. It is also a good idea to check the “Freeze Rotation” under “Constraints” and tick the X and Z boxes; doing so will prevent the agent from falling over when moving. This is illustrated in Figure 26.

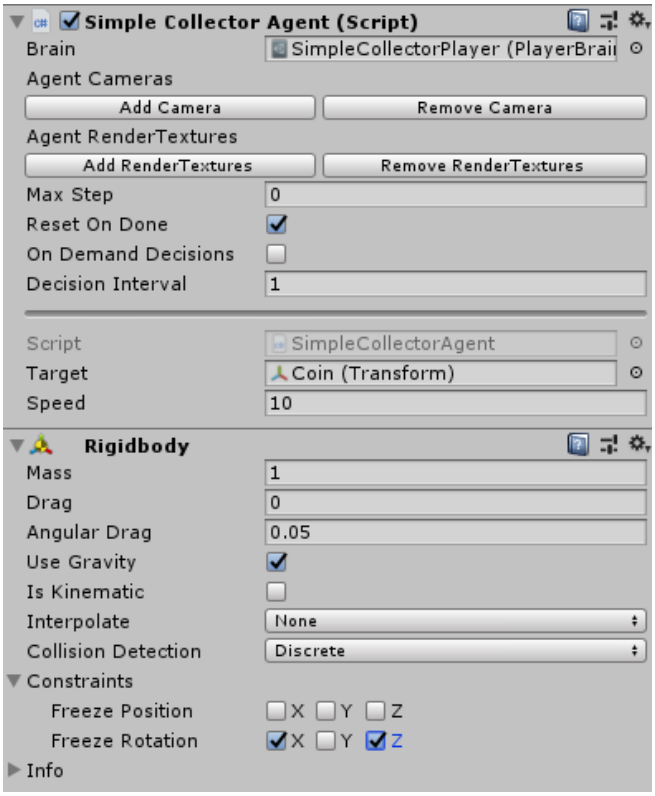


Figure 26. The Rigidbody and SimpleCollectorAgent components inside the Collector-Agent GameObject

The last feature the environment needs is a platform for the player to walk on as well as a target objective. The ground will be a simple “Plane” GameObject with only a collider on it and the objective will be a “Cylinder” GameObject, which will be scaled on the y-axis to 0.1 to look like a coin. The plane GameObject will be renamed “Ground” and the cylinder GameObject will be renamed “Coin”. To give the coin and ground some flavor, they should have different materials to be easily distinguished from the environment. The ground will have white material with a grid on it and the coin will be given a yellow material as illustrated in Figure 27.

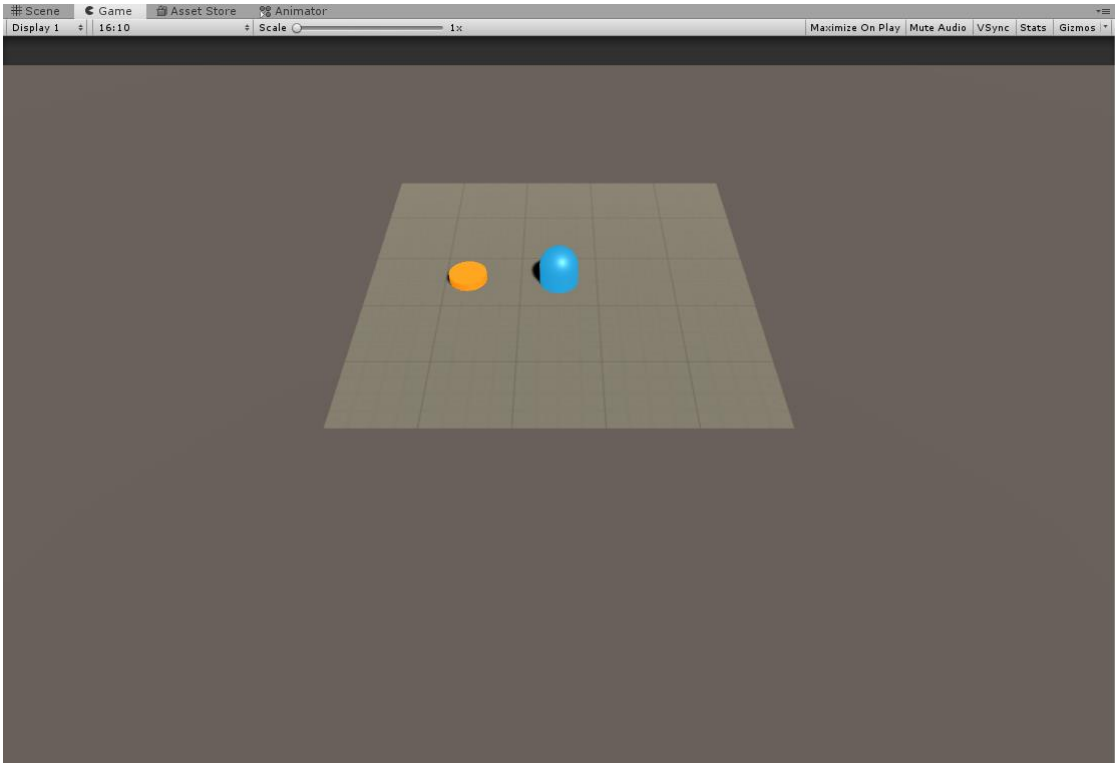


Figure 27. The current Game view rendered from the Camera's location

Before moving on to scripting the logic for the Agent, it is wise to move the player GameObject as well as the coin to a better position, since they are both clipping through the ground. The player's position will be moved to [0, 1, 0], which means that the x- and z-axis of the GameObject are zero and the y-axis of the GameObject is one. For the coin the position will be [-8, 1, 0] and the rotation will be set to [0, 0, 90] which will rotate it on the z-axis by 90 degrees. A logical idea is to also make the playing ground slightly larger to give the agent some challenge, the ground's scale will be set top [2, 2, 2] as seen in Figure 28.

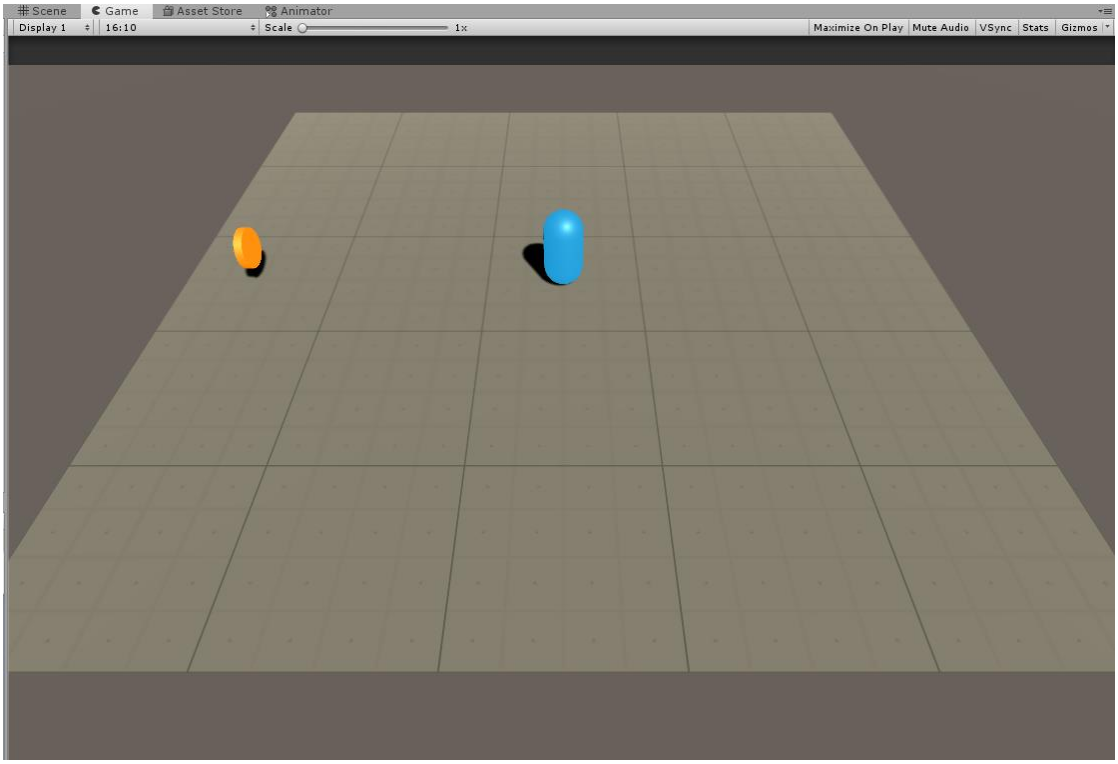


Figure 28. Environment set up for the agent

4.4.3 Controlling the agent

Now that the environment is set up, the next step is to create the logic for the agent. The main variables the agent needs are a reference to the Rigidbody component it-self, the speed multiplier, the movement direction as a vector as well as a reference to the target. To get the Rigidbody reference, it is going to be acquired from the "Start" method of the script using the "GetComponent" method. The "Start" method will be called as the game is launched before the first "FixedUpdate" as illustraded in Figure 29.

```
public class SimpleCollectorAgent : Agent {

    public Transform target;
    public float speed = 10;

    private Rigidbody rb;
    private Vector3 moveDir;

    0 references
    private void Start()
    {
        rb = GetComponent<Rigidbody>();
    }
```

Figure 29. SimpleCollectorAgent variables and" Start" method

Next up will be the “AgentReset” method that will be called once the agent is reset using the “Agent.Done()” command. Inside the method, it is going to have a few simple lines that will randomize the position of the coin (target), zero the agent’s velocity as well as respawn it back on top of the ground GameObject illustrated in Figure 30.

```
24 references
public override void AgentReset()
{
    // Zero velocity and respawn back in the middle
    rb.velocity = Vector3.zero;
    transform.position = new Vector3(0, 1, 0);

    // Move the target to a new random spot
    target.position = new Vector3(Random.value * 16 - 8, 0.5f, Random.value * 16 - 8);
}
```

Figure 30. AgentReset method

After configuring the resetting of the agent, it is good to create the logic for the agent’s movement. The logic will be written inside the “AgentAction” method, which handles the agent’s actions. The “AgentAction” method comes with a “vectorAction” property, which can be used to transfer inputs inside the method. For this agent it needs two vector actions, which corresponds to forward/backward and left/right movement. The input actions have not yet been implemented to the player brain but the logic for the movement can already be made. As shown in Figure 31, the agent’s Rigidbody velocity will be changed if the corresponding vector action inputs are pressed, the agent will also rotate towards the movement direction.

```
23 references
public override void AgentAction(float[] vectorAction, string textAction)
{
    // vectorAction[0] = Left/Right
    // vectorAction[1] = Forward/Backward

    Vector3 controlSignal = Vector3.zero;
    moveDir.x = vectorAction[0] * speed;
    moveDir.y = rb.velocity.y;
    moveDir.z = vectorAction[1] * speed;

    transform.LookAt(transform.position + new Vector3(moveDir.x, 0, moveDir.z));
    rb.velocity = moveDir;
}
```

Figure 31. AgentAction method

The last method that needs to be configured is the “CollectObservations” method, which will handle the data collection of the agent. The observations that the agent wants to collect are its current position, the position of the target as well as the velocity x and z of the agent as seen in Figure 32.

```
24 references
public override void CollectObservations()
{
    // Agent and Target positions
    AddVectorObs(target.position);
    AddVectorObs(this.transform.position);

    // Agent velocity
    AddVectorObs(rb.velocity.x);
    AddVectorObs(rb.velocity.z);
}
```

Figure 32. CollectObservations method

After the logic has been created, the brains of the agent need configuration as well. When selecting the “SimpleCollectorPlayer” in the project window, the brain’s details are displayed in the inspector view. The Space Size parameter under Vector Observations in the brain’s details is required to be the same size as the amount of observations inside the “CollectObservations” method. For this parameter, it will be set to eight, since the “AddVectorObs(target.position)” and “AddVectorObs(transform.position” both collect three observations (x-, y- and z-axes) each, and the velocity observations collect one observation each. The “Space Type” parameter must be set from “Discrete” to “Continuous” for the continuous movement of the agent; after setting it, a new parameter will appear at the bottom of the component called “Key Continuous Player Actions”. The “Key Continuous Player Actions” needs a size of four and each element will be given a reference to a key as well as the value of said key as can be seen in Figure 33.

Now if the “Play” button is pressed, the agent can be controlled by the player using the WASD keys to move around.

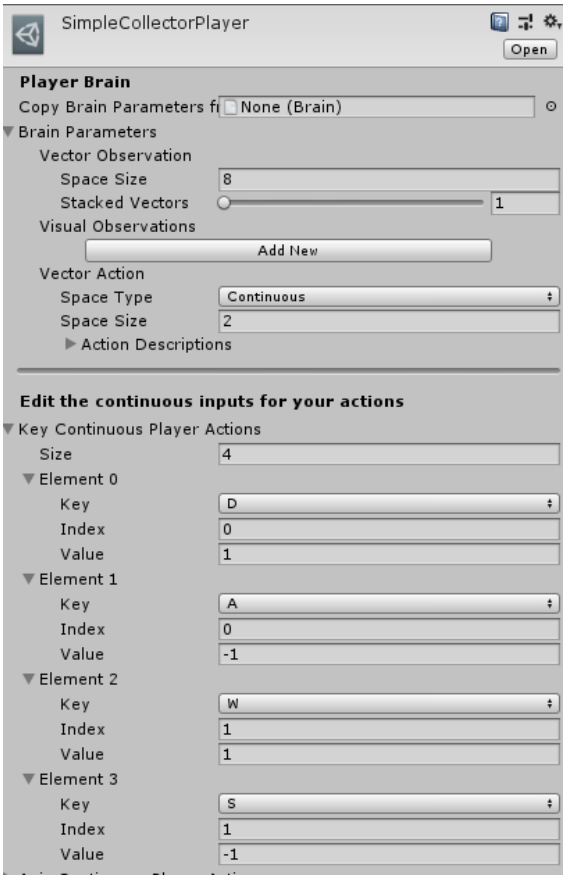


Figure 33. SimpleCollectorPlayer brain after adding necessary information

4.4.4 Rewarding the Agent

Before the training process for the agent can begin, the agent will need to know what rewards it and what does not. This can be achieved using the “AddReward()” and “SetReward()” function. When the agent picks up the coin, it should be rewarded doing so and if the agent falls off the platform, it will get no reward.

For the logic of picking up the coin, the coins collider component should be changed into a trigger collider. To achieve this the coin needs to be selected in the Hierarchy window after which the “Capsule Collider” components’ “IsTrigger” property will need to be checked in the Inspector window as seen in Figure 34.

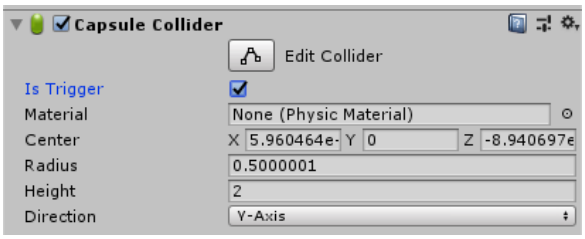


Figure 34. Capsule Collider "IsTrigger" property checked

Unity has a simple way for checking if a collider has hit another collider or trigger by using the “OnTriggerEnter()” method. The method will be called once the agent has touched the target object and after collecting the agent will be rewarded after which the “AgentReset()” method will be called using “Done()” to relocate the coin to a new spot. Figure 35 illustrates this.

```
0 references
private void OnTriggerEnter(Collider other)
{
    if (other.gameObject == target.gameObject)
    {
        SetReward(1f);
        Done();
    }
}
```

Figure 35. OnTriggerEnter method

The best way to punish one’s agent is to make it reset every time the agent falls from the platform. To do this it can be added to the “AgentAction” method checking if the agent’s transform position on the y-axis goes under zero; if it does, the agent will be reset back into the platform and the coin will be relocated as well as illustrated in Figure 36. In the “AgentReset” method it would also wise to also add a check if the player fell, because it would be logical for the agent not to spawn back in the middle of the ground if it picks up a coin as seen in Figure 37.

```
// Fell off platform
if (this.transform.position.y < 0)
{
    Done();
}
```

Figure 36. Fall check inside the AgentAction method

```
// If the agent fell of the platform
if (this.transform.position.y < 0)
{
    // Zero velocity and respawn back in the middle
    rb.velocity = Vector3.zero;
    transform.position = new Vector3(0, 1, 0);
}
```

Figure 37. Fall check inside the AgentReset method

4.4.5 Training the Agent

To start the training process, the agent's brain will need to be switched to the "SimpleCollectorLearning" brain and modify its "Vector Observation/Space Size" to eight, the "Vector Action/Space Type" to continuous as well as the "Vector Action/Space Size" to two as seen in Figure 38.

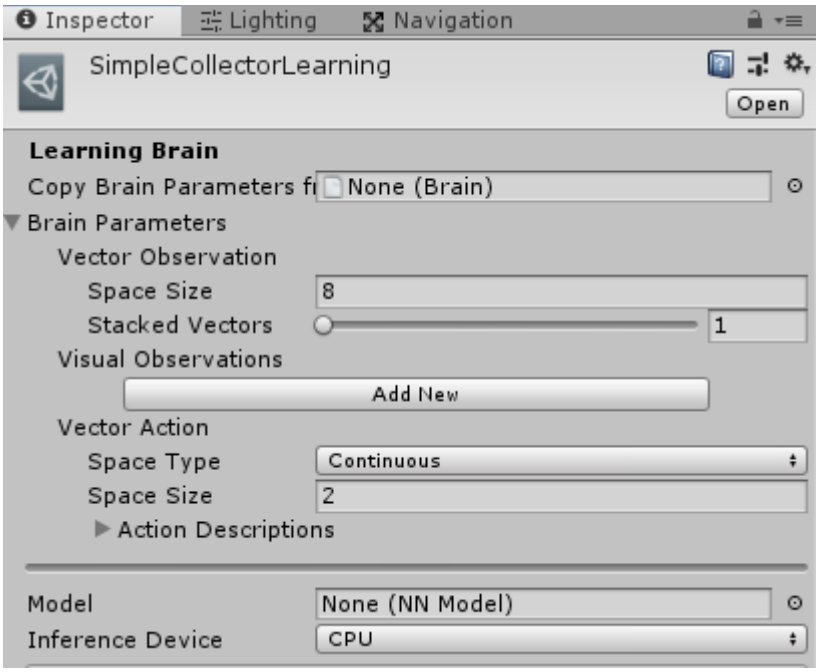


Figure 38. The learning brains data

The learning brain will need to be modified inside the academy GameObject as well as ticking the "Control" box beside the brain property slot as shown in Figure 39.

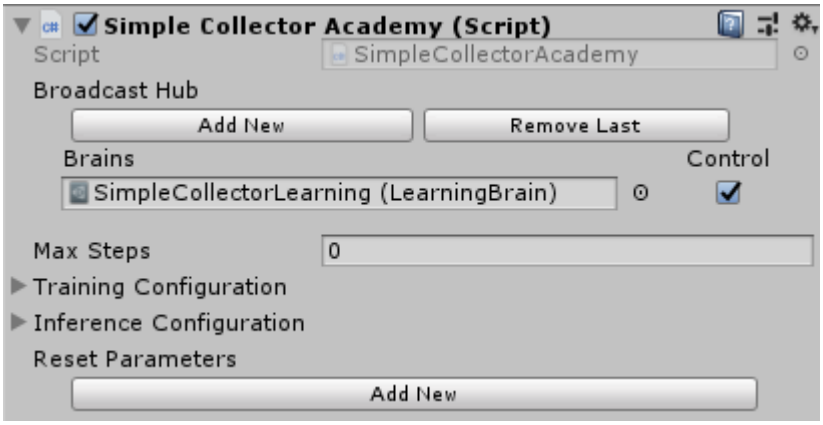


Figure 39. academy GameObject

Now using the Anaconda Prompt the training process can be started by activating the Conda learning environment created before and typing this command *"mlagents-learn config/trainer_config.yaml –run-id=SimpleCollectorRun01 –train"* inside the Anaconda Prompt once the environment has been activated. After typing the command in the Anaconda Prompt and hitting the "Play" button inside Unity, the agent will now start training itself in the new environment.

Training with only one agent can be slightly more time consuming; as was seen in the 3DBallLearning environment, there were multiple platforms that tried to balance the ball. To achieve the same effect here the environment can be made into a prefab after which it can be duplicated into the scene thus training more agents at the same time. To create the prefab, the agent, coin and ground GameObjects will need to be added into an empty GameObject which will be named "TrainingGround". Once the previously mentioned GameObjects have been transferred inside the new "TrainingGround" GameObject as shown in Figure 40, the GameObject can now be made into a prefab by dragging it inside the "SimpleCollector" folder in the Project Window. To make things clearer for developers, it is recommended to create a separate "Prefabs" folder inside the "SimpleCollector" folder as illustrated in Figure 41.

Figure 40. The new TrainingGround GameObject with the necessary GameObjects as children

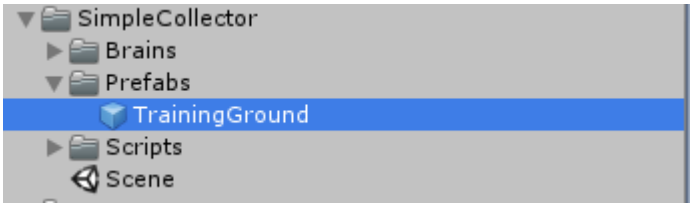


Figure 41. TrainingGround as a prefab

Now that the prefab has been created, it is now possible to easily duplicate the prefab inside the hierarchy and add more training grounds into the environment. Before the training can start, it is now mandatory to make some modifications to the scripts. Every line that contains “transform.position” should now be changed to “transform.localPosition”, since the agents move inside the parent GameObject so the agent’s position should be localized to that parent GameObjects position as seen in Figure 42.

```
// Agent and Target positions
AddVectorObs(target.localPosition);
AddVectorObs(this.transform.localPosition);
```

Figure 42. Example of the localPosition change

After retyping the command inside the Anaconda Prompt and hitting the “Play” button inside Unity, all the agents are training in their own corresponding training grounds shown in Figure 43.

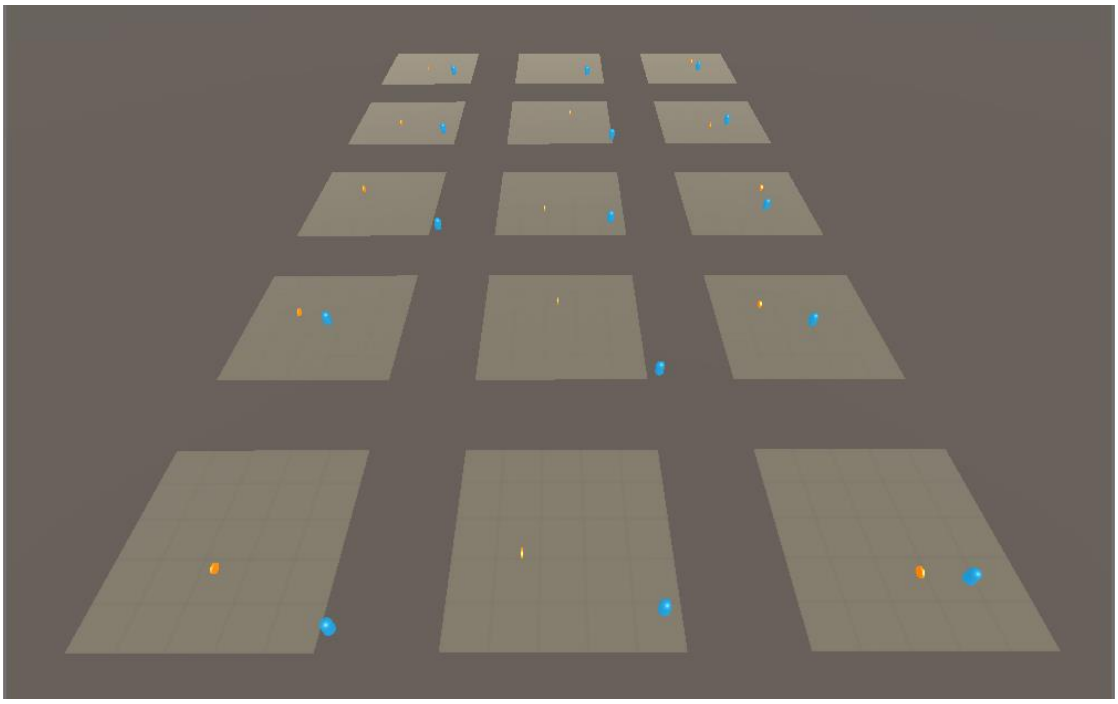


Figure 43. Agents training in their own training grounds

4.4.6 Training Data

The training data can be once again found inside TensorBoard. For this, there will be two different training data, first being the data trained with only one agent and the second will be the one that trained with fifteen agents. Both training sessions lasted 13 000 steps and based on the data, the one with more training agents were able to train at a faster pace and learned to play the game faster than the training session with only one agent.

The training data shows how the second run with fifteen training grounds helped the agents to learn at a more stable and quicker rate compared to the one with only one training ground. Training with multiple agents can take more time; for example, the training session with fifteen training grounds took two minutes to complete the training while the other only took forty-two seconds to complete the training. Overall the value for having multiple training grounds speeds up the training process, best way to analyze this is by looking at the “Cumulative Reward” graph as shown in Figure 44. The graph indicates that the brain that trained with fifteen training grounds

accumulated better rewards than the brain with only one training ground. See Appendix 6 and 7 for the full TensorBoard graph output.

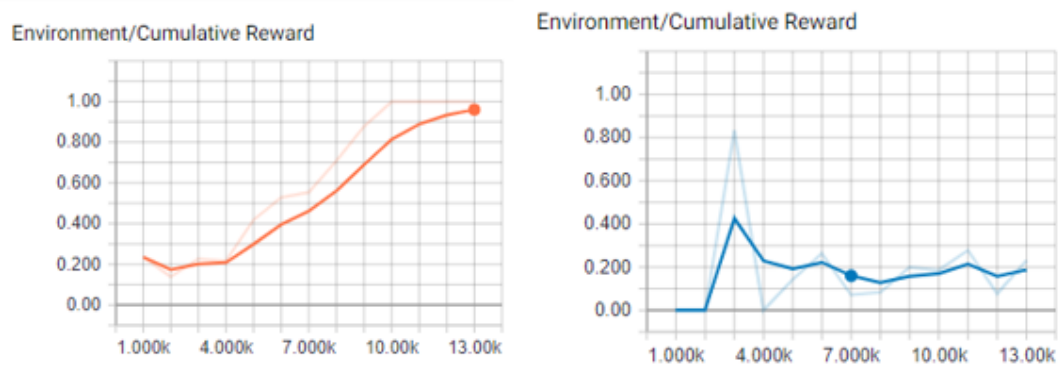


Figure 44. Cumulative Reward data for both training sessions. Orange trained with fifteen agents while blue trained with one agent

Another good way to compare the two learned models is to implement them into the project and create a duplicate learning brain that hosts the other trained model. A nifty addition to this is to include a coin counter that counts the amount of the coins that both agents collect and measure how many coins they collect in one minute. Both agents get their own platform with a Text GameObject above their platforms that display the amount of coins they collect. The “SimpleCollectorAgent” script needs a reference to the text element to change the amount of coins the agents have acquired; Figure 45 shows that the agents also need a new private integer that counts the amount of current coins.

```
0 references
private void OnTriggerStay(Collider other)
{
    if (other.gameObject == target.gameObject)
    {
        // Add a coin to and change the text inside the game view
        amountOfCoins++;
        coinText.text = "Coins: " + amountOfCoins.ToString();

        SetReward(1f);
        Done();
    }
}
```

Figure 45. Adding the coin counter

Once the corresponding text elements have been referenced to the agents, hitting the “Play” button will now show how the agents interact with the environment using their trained models. Figure 46 shows the result of how many coins each agent acquired in exactly one minute of gameplay. From that gameplay session, the one that had fifteen training grounds during training was able to collect 240 coins while the one with only one training ground during training was only able to collect 13 coins.

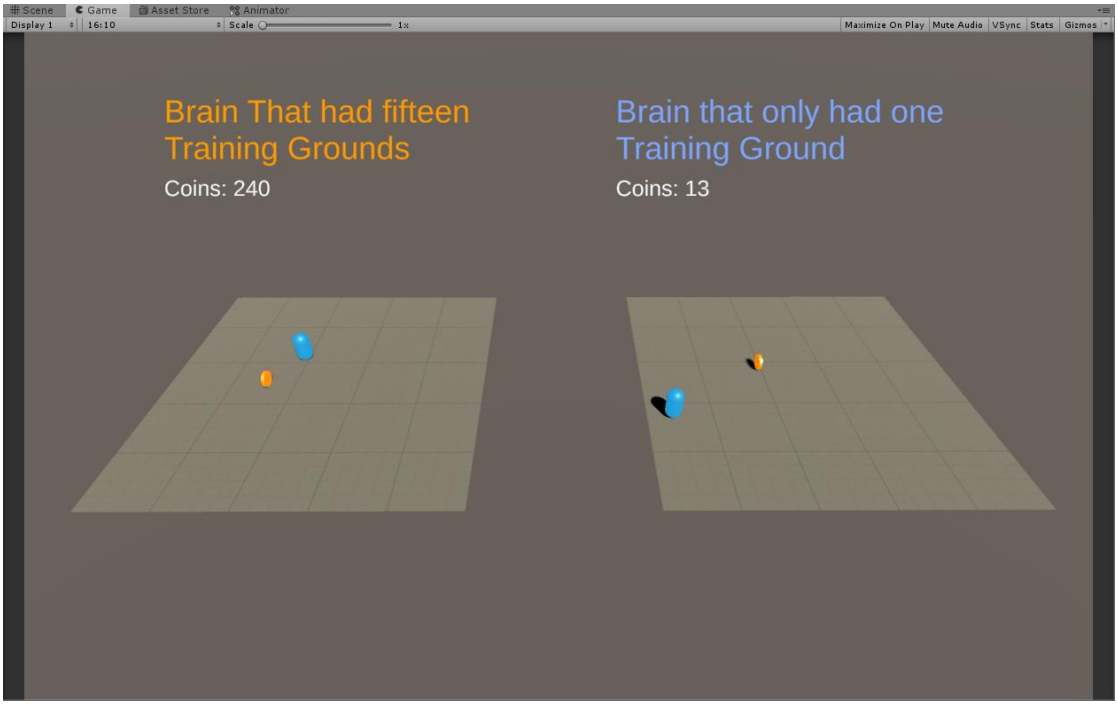


Figure 46. One minute of gameplay using the trained models

4.5 Creating the AI using NavMeshAgent

Before the Unity ML-Agents toolkit AI and the traditional Unity Navigation AI can be compared, the latter will need to be created. To start off, the Unity ML-Agents’ environment training ground prefab needs to be duplicated and named “TrainingGround-NavMesh”. The new prefab is opened and the “CollectorAgent” GameObject is located in the hierarchy view, after which the “SimpleCollectorAgent” script can be removed via the inspector window, since the script is only required when using Unity

ML-Agents. To create a Unity NavMeshAgent, the GameObject will need a component called “NavMeshAgent”. The “NavMeshAgent” component handles all the required navigational data an AI will need including steering, obstacle avoidance and path finding. The speed value of “NavMeshAgent” components will be changed to fifteen to match the speed with the Machine Learning agents, which can be seen in Figure 47.

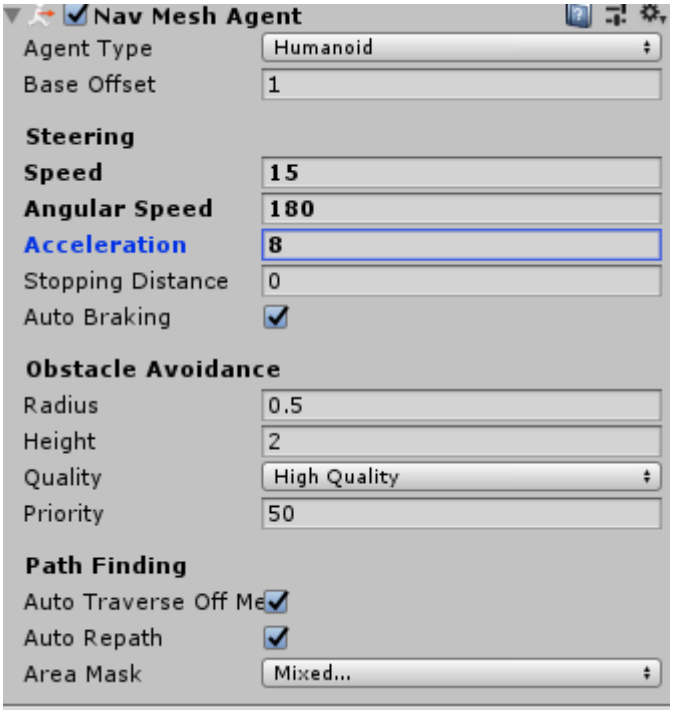


Figure 47. Unity NavMeshAgent component

The NavMeshAgent will require a separate script component that will generally handle the destination logic of the agent; this script will be named “SimpleCollectorNavMesh”. Just like the Unity ML-Agent script that was created, the NavMeshAgent will need a reference to the target location as well as a reference to the coin counter which will be used when determining the results between these agents. Unlike the ML-Agent, the “SimpleCollectorNavMesh” script requires a reference to the NavMeshAgent component attached to the AI Agent. The variables for this new script can be seen in Figure 48.

```
public class SimpleCollectorNavMesh : MonoBehaviour
{
    public Transform target;
    public TextMeshProUGUI coinText;

    private NavMeshAgent agent;
    private int amountOfCoins;

    0 references
    void Start()
    {
        agent = GetComponent<NavMeshAgent>();
    }
```

Figure 48. Variables and reference to the NavMeshAgent in the Start method

The last mandatory methods the “SimpleCollectorNavMesh” script needs are the “OnTriggerStay” method that will check if the agent has collected a coin as well as a simple “ResetCoin” method that will randomly place the coin to a different location as shown in Figure 49.

```
1 reference
private void ResetCoin()
{
    target.localPosition = new Vector3(Random.value * 8 - 4, 0.5f, Random.value * 8 - 4);
}

0 references
private void OnTriggerStay(Collider other)
{
    if (other.gameObject == target.gameObject)
    {
        ResetCoin();
        amountOfCoins++;
        coinText.text = "Coins: " + amountOfCoins.ToString();
    }
}
```

Figure 49. OnTriggerStay and ResetCoin methods for SimpleCollectorNavMesh script

The NavMeshAgent does not need a method for resetting the agent’s position, since theoretically the agent cannot fall off the platform, because of the NavMesh that is baked onto the platform. When baking NavMesh, it will only be baked on top of meshes marked as “Navigation Static” in the inspector window as shown in Figure 50.

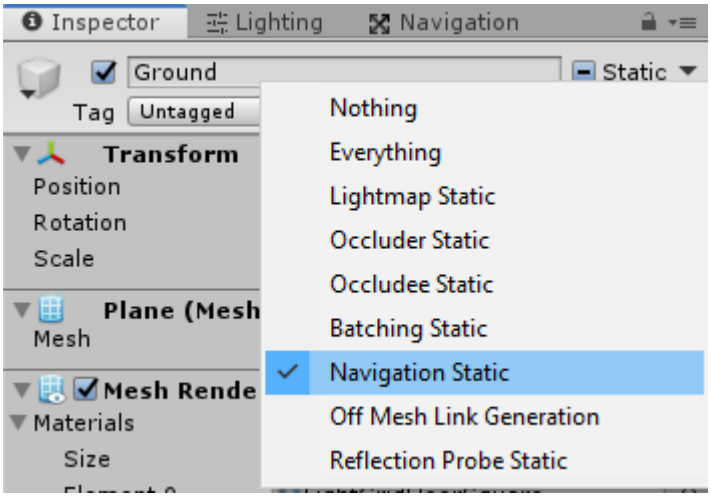


Figure 50. Navigation Static before baking NavMesh

The Navigation window, which can be located on top of Unity Editor toolbar in “Window/AI/Navigation”, displays everything related to NavMeshAgents, NavMeshObstacles and NavMesh baking. In the Navigation window, one can modify the NavMeshAgent radius and height as well as create new agents with different height and radius settings. In the “Bake” tab in the Navigation window one can modify the different settings related to baking the NavMesh. Leaving these settings in their default values can be a good idea. To bake the NavMesh, simply pressing the “Bake” button will start the process; and since the current environment is simple, it is done in a mere second. The baked NavMesh as well as the NavMesh baking settings can be seen in Figure 51.

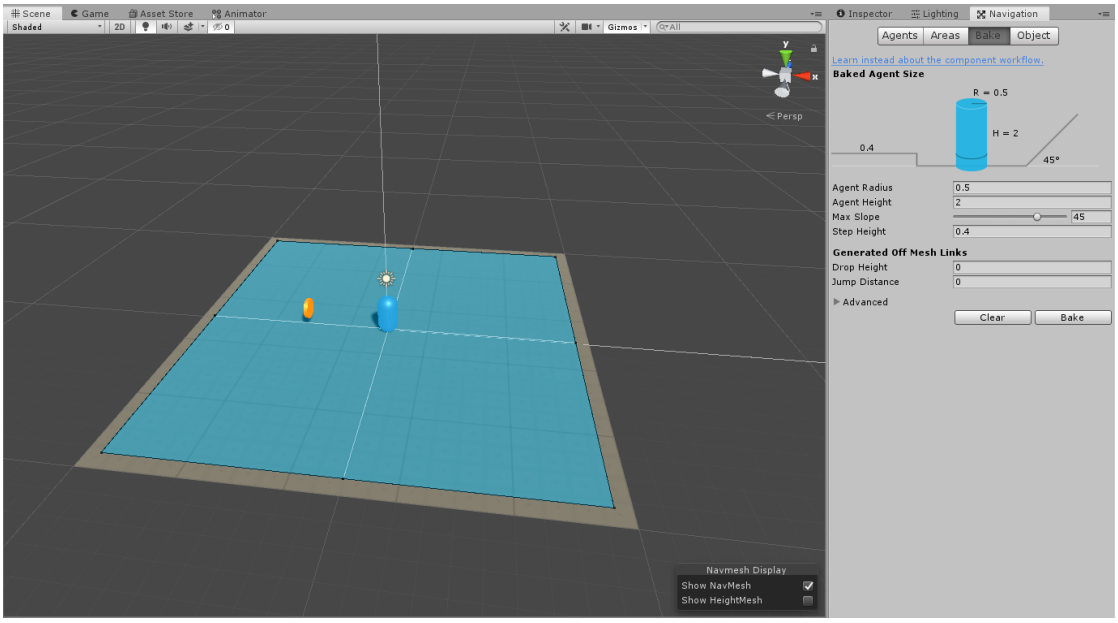


Figure 51. Baked NavMesh

The agent will need one more line of code, which will inform the NavMeshAgent what its destination is. This is simply done by giving the NavMeshAgent a reference of the position of the coin GameObject, which will be done inside of the “Start” method as well as inside the “ResetCoin” method as shown in Figure 52. After hitting the “Play” button, the AI will now collect coins on the training ground unable to fall off the platform and always finding the best route for the target. After one minute of gameplay, the agent was able to collect 70 coins as shown in Figure 53.

```
0 references
void Start()
{
    agent = GetComponent<NavMeshAgent>();
    agent.SetDestination(target.localPosition);
}

1 reference
private void ResetCoin()
{
    target.localPosition = new Vector3(Random.value * 8 - 4, 0.5f, Random.value * 8 - 4);
    agent.SetDestination(target.localPosition);
}
```

Figure 52. Giving the NavMeshAgent its destination position

Figure 53. One minute of gameplay with the NavMeshAgent

5 Results

Comparing the Unity ML-Agent and the traditional Unity Navigation AI can be difficult since the Unity Navigation AI did not give much relevant data except for the coins collected. The training data for the ML-Agent simply shows the learning process of the brain that can be used to determine how clever the machine-learnt agent is at figuring out the core game idea, while the Unity NavMeshAgent knows from the first frame of the game what it needs to accomplish in the game.

The Unity NavMeshAgent is a handwritten AI that will never fall off the platform, making it autonomous to the surrounding environment; however, it is always required to have a baked NavMesh under its feet, which makes it different from the Unity ML-Agent. On the other hand, the Unity ML-Agent does not know anything about the game before it has been trained, making it fall off the platform dozens of times before learning the core concept of the game. Because of this, the Unity ML-Agent is more independent and trains itself the best methods and mechanics to use in the game compared to the traditional Unity Navigation AI, which automatically knows what to do without requiring any training.

The best way to compare the results of both systems is to add the side-by-side in the coin collector test. In this test, the Unity Navigation AI will compete in collecting coins beside the Unity ML-Agent brain that has had fifteen different training grounds. Both contestants had two minutes to collect as many coins as possible, and the results for this test conclude that the even though the traditional Unity AI was simpler and easier to create, the Unity ML-Agent was able to collect over double the amount of coins. Unity NavMesh Agent collected 153 coins compared to the whopping 439 coins collected by the self-learnt Unity ML-Agent as seen in Figure 54.

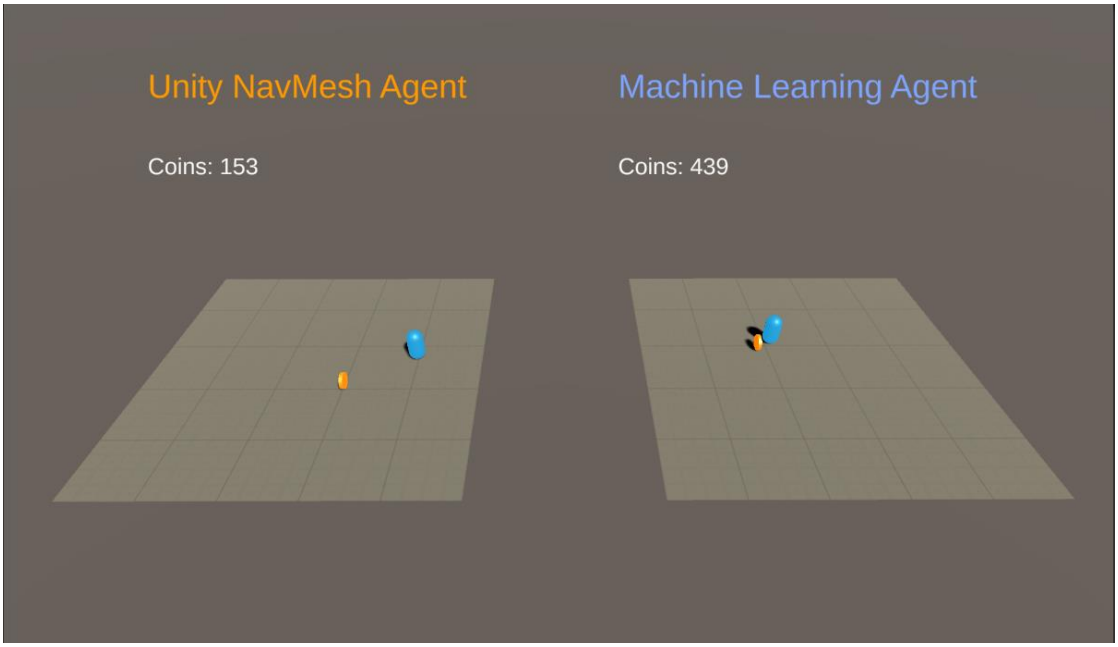


Figure 54. The result for two minutes of gameplay between the NavMeshAgent and ML-Agent

6 Conclusion

The Unity Machine Learning Agents compare to traditional NavMesh AI in dozens of different ways. The main comparisons are that the Unity Machine Learning toolkit can be a bit trickier to set up than the NavMeshAgents inside Unity, but the output of using Machine Learning is worth it, since it allows developers to create functional AI systems without needing to teach them the core gameplay methods all that much. The test that was carried out in these environments was simple for both agent types, however, creating a more complex AI using Unity NavMesh system can be trickier and more challenging for developers. Thus, using Machine Learning allows developers to let the agents teach on their own without having to worry about problems regarding Navigation Mesh.

With Machine Learning, developers can more easily integrate smarter and advanced super-human AI into their games than creating it from scratch using traditional hard-coded AI logic. When comparing both the Machine Learning AI and the NavMesh AI, it was clearly indicated how intelligent the former was to the latter.

References

Budek, K. & Osiński, B. 2018. What is reinforcement learning? The complete guide. July 5, 2018. Accessed on 9 August 2019. Retrieved from https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/

Cluster Analysis Example Image. 2019. Wikipedia. Accessed on 9 August 2019. Retrieved from https://en.wikipedia.org/wiki/Cluster_analysis

Dillet, R. 2018. Unity CEO says half of all games are built on Unity. September 2018. Accessed on 9 August 2019. Retrieved from https://techcrunch.com/2018/09/05/unity-ceo-says-half-of-all-games-are-built-on-unity/?guccounter=1&guce_referrer_us=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_cs=647JIDfu7if2j6IVnaKvzA

Dwivedi, D. 2018. Machine Learning For Beginners. Towards Data Science. Accessed on 9 August 2019. Retrieved from https://towardsdatascience.com/machine-learning-for-beginners-d247a9420dab

Esposito, M., Bheemaiah, K., & Tse, T. 2017. What is machine learning? Accessed on 9 August 2019. Retrieved from https://theconversation.com/what-is-machine-learning-76759

Juliani, A., Berges, V., Vckay, E., Gao, Y., Henry, H., Mattar, M. & Lange, D. 2018. Unity: A General Platform for Intelligent Agents. Accessed on 12 August 2019. Retrieved from https://arxiv.org/abs/1809.02627

Middleton, Z. 2002. Case History. The Evolution of Artificial Intelligence in Computer Games. Accessed on 6 August 2019. Retrieved from https://web.stanford.edu/group/htgg/cgi-bin/drupal/sites/default/files2/zmiddleton_2002_1.pdf

Salian, I. 2018. SuperVize Me. What's the Difference between Superviced, Unsupervised, Semi-Supervised and Reinforcement Learning? Accessed on 9 August 2019. Retrieved from https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/

Simonite, T. 2018. Can bots outwit human in one of the biggest esport games? Accessed on 9 August 2019. Retrieved from https://www.wired.com/story/can-bots-outwit-humans-in-one-of-the-biggest-esports-games/

Creating and Using Scripts. 2019. Unity Documentation. Accessed on 6 August 2019. Retrieved from https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html

Navigation and Pathfinding. 2019. Unity Documentation. Accessed on 2 September 2019. Retrieved from https://docs.unity3d.com/Manual/Navigation.html

Scenes. 2019. Unity Documentation. Accessed on 6 August 2019. Retrieved from https://docs.unity3d.com/Manual/CreatingScenes.html

Unity Licence Agreement 2019. Licence Agreement from Unity's website. Accessed on 6 August 2019. Retrieved from https.//unity3d.com/legal/terms-of-service

Get to grips on the latest AI power in Unity. 2019. Unity Website. Accessed on 12 August 2019. Retrieved from https.//unity3d.com/how-to/unity-machine-learning-agents#how-it-works

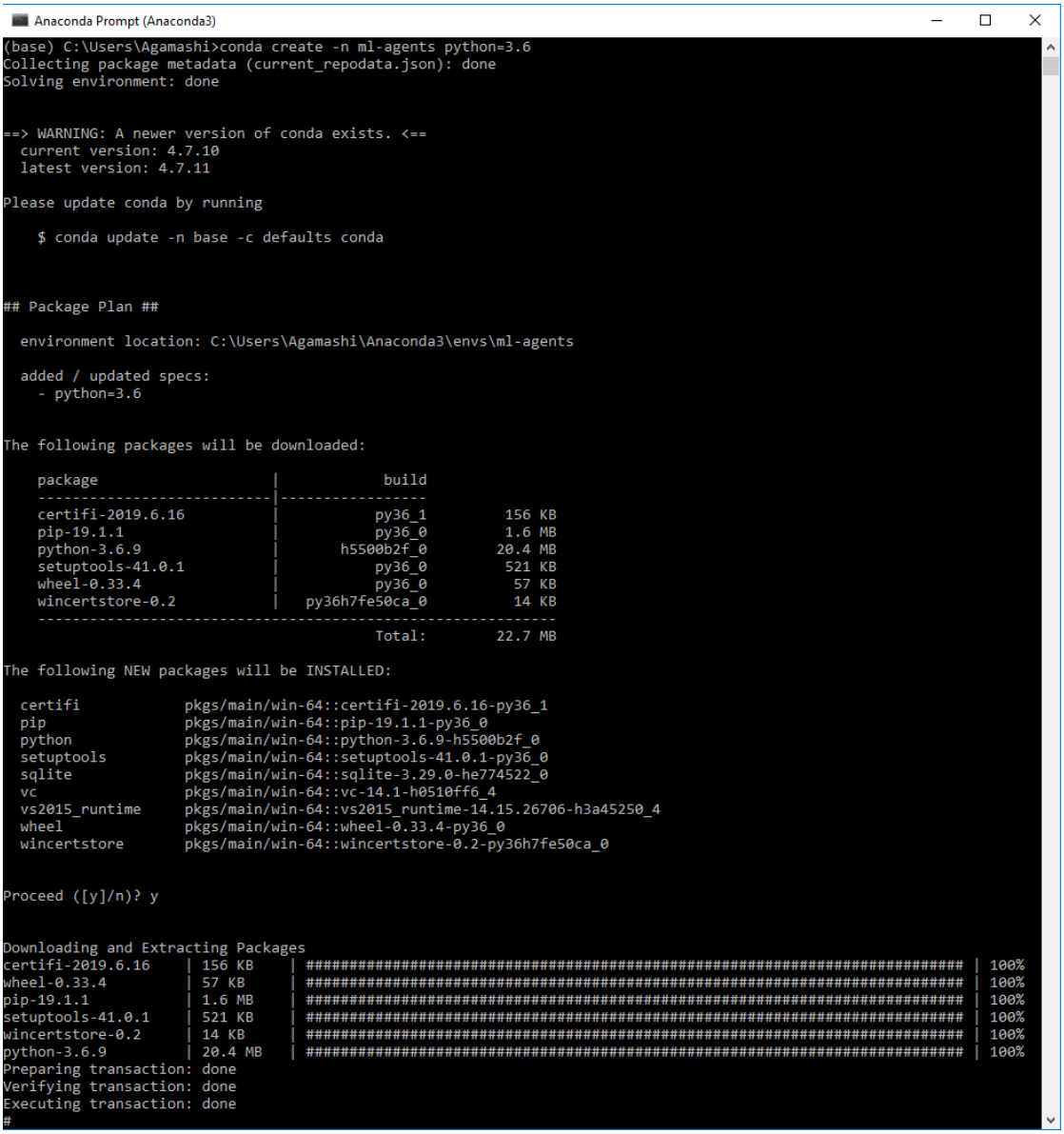
Unity. 2019. Page from Unity's website. Accessed on 6 August 2019. Retrieved from https.//unity3d.com/unity

Wiederhold. G, McCarthy, J. & Feigenbaum, E. N.d. Professor Arthur Samuel. Accessed on 9 August 2019. Retrieved from https.//cs.stanford.edu/memoriam/professor-arthur-samuel

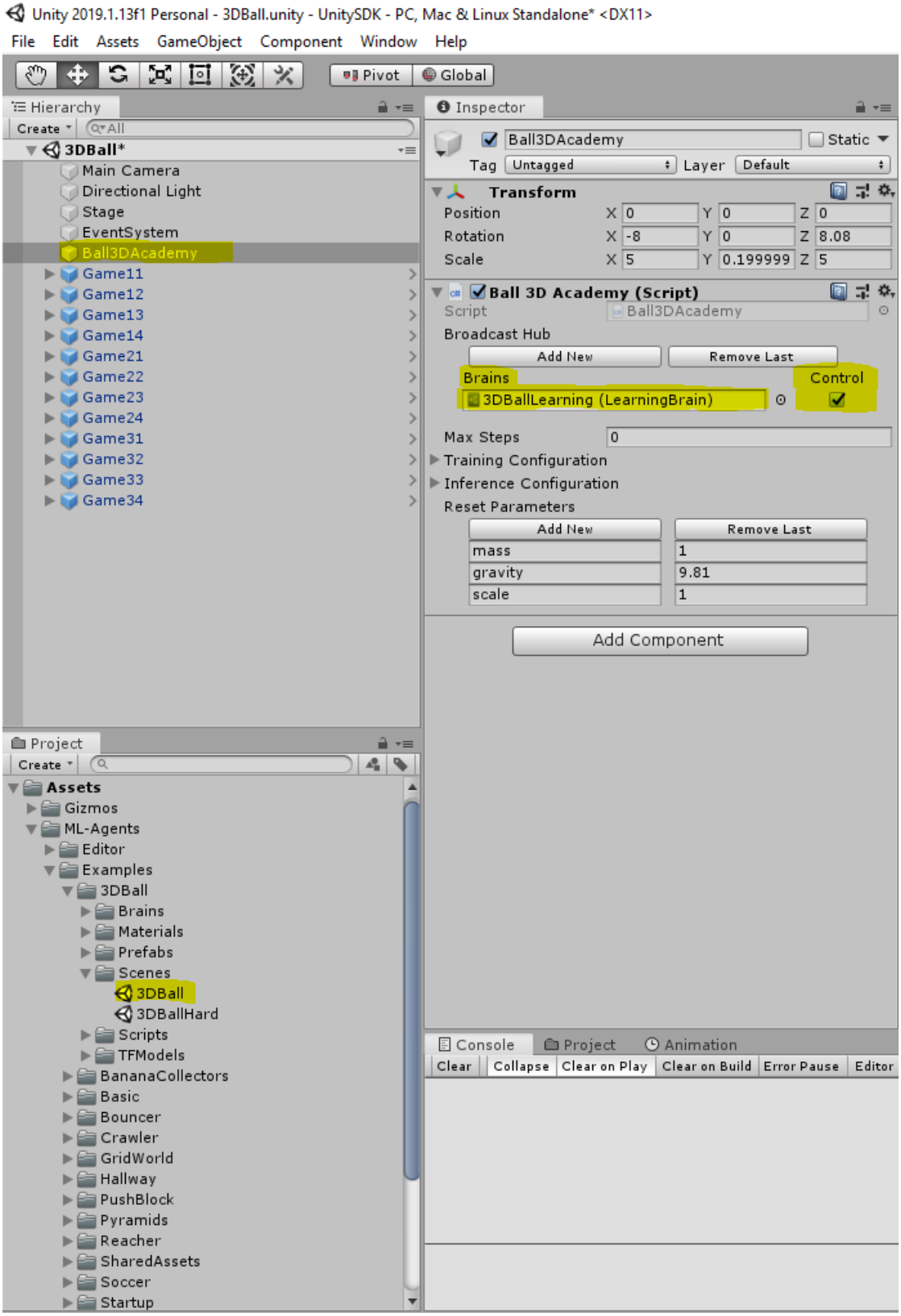
Yegulalp, S. 18.6.2019. What is TensorFlow? The machine learning library explained. Accessed on 12 August 2019. Retrieved From https.//www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html

Appendices

Appendix 1. Full console view of the creation of a Conda Environment

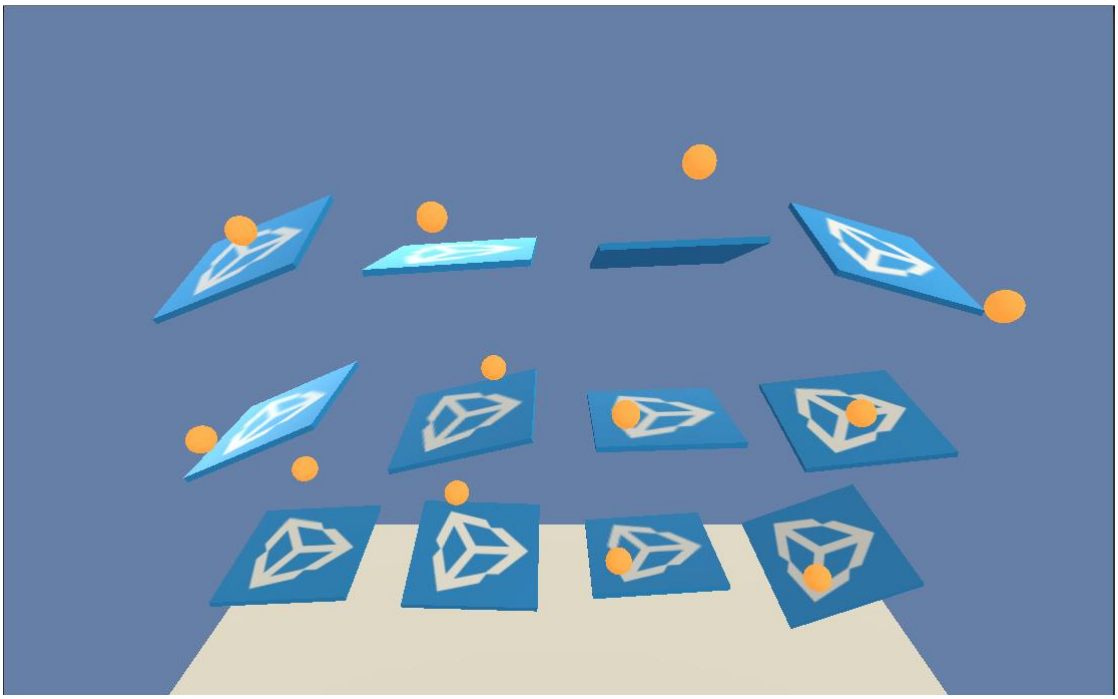


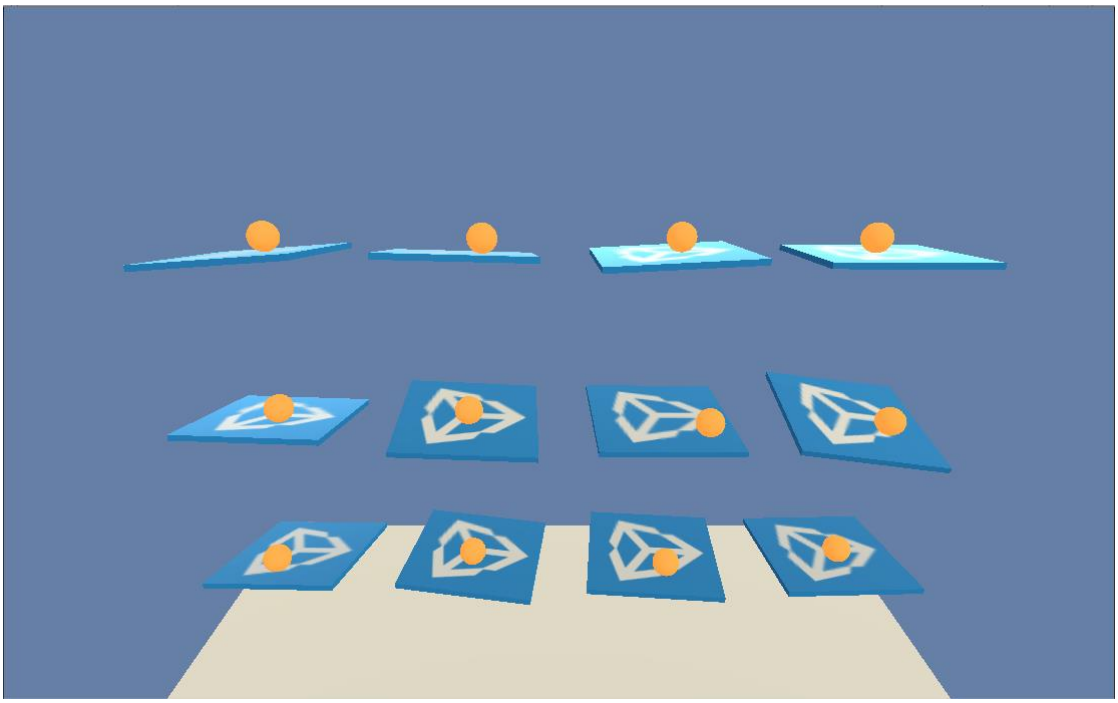
Appendix 2. Full editor view of the 3DBall example



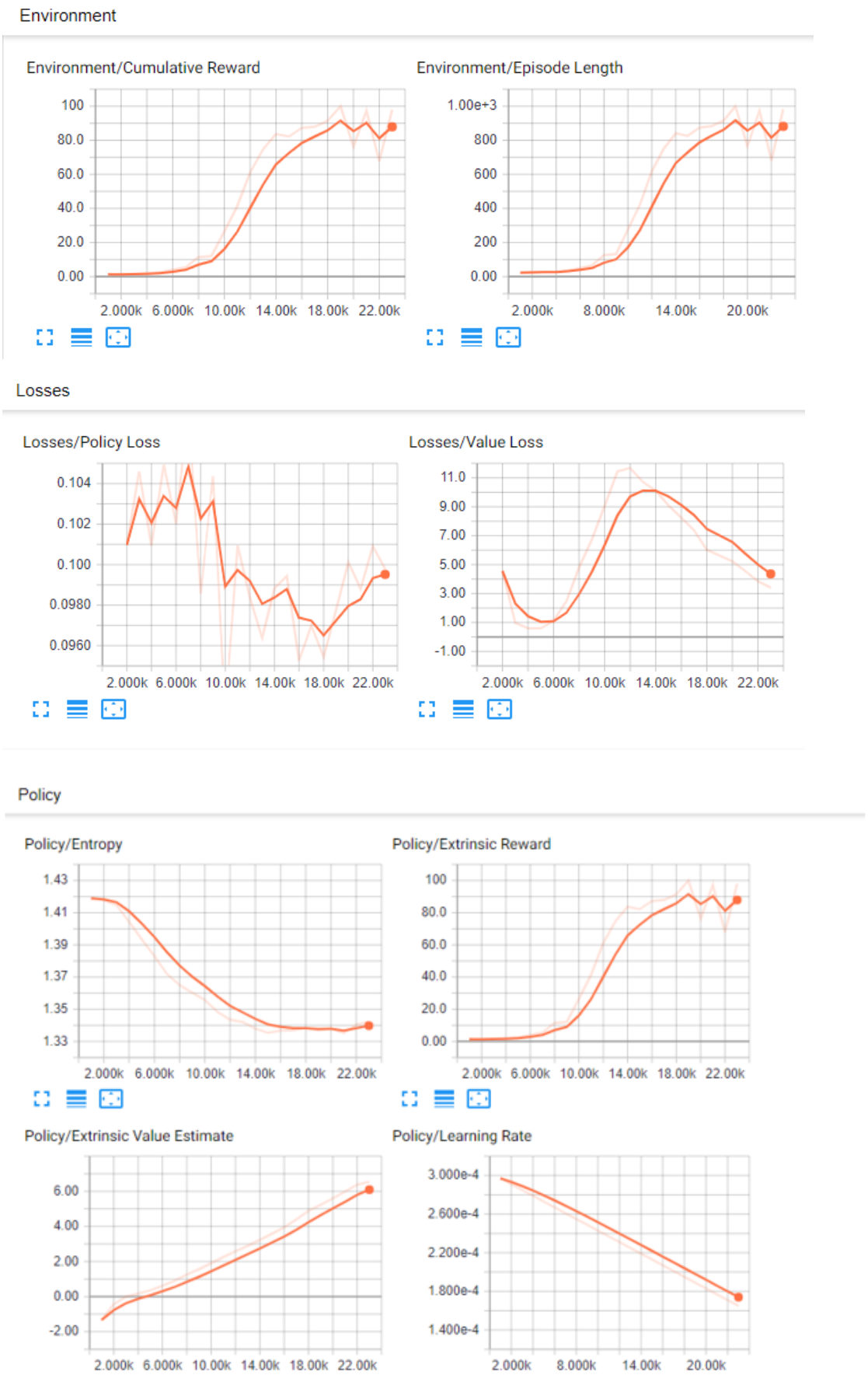
Appendix 3. Console view after typing the appropriate command in order to start training the brain.

Appendix 4. Training the 3DBall brain with Reinforcement Learning. First screenshot represents the point where the brain has not trained to balance the ball on the platform and the last picture represents the point where the brain has trained to balance the ball after three minutes of training at one-hundred times the regular speed, which is five hours of training done in three minues.





Appendix 5. 3DBallLearning training data via TensorBoard.



Appendix 6. Training Data for SimpleCollector with only one Training Ground in the environment

Appendix 7. Training Data for SimpleCollector with fifteen Training Grounds in the environment