Adapting Machine Learning Models in the Unity Game engine to Evaluate Performance on Different Hardware Limitations

Rashid Hafez

August 3, 2020

Declaration

I, Rashid Hafez confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: Rashid Hafez

Date: August 3, 2020

Abstract

Machine learning is an approach to programming where instead of the writer using hard coding techniques, the program learns from data and adapts. Nowadays, games are ran on multiple different platforms and devices, such as mobiles, laptops computers and other handheld devices such as the Nintendo Switch. This project looks at running machine learning algorithms in the unity game engine and evaluating its performance on different devices such as the Nintendo Switch.

Contents

1 Introduction

1.1 Aims

This research project aims to develop and implement an artificial agent in the unity game engine using ML algorithms such as neural networks and reinforcement learning to complete a goal in the game. Then pushing the system which the algorithm is implemented in to its limits in order to measure performance. One way to accomplish this is by testing multiple instances of the algorithm at the same time on a singular machine then port the code onto different machines specifically handheld devices such as the Nintendo Switch and mobile devices. In hindsight for the future, the final aim for this project is to use this work to optimise future algorithms or procedures to improve current AI for games.

Since models and resources are sparse, the game will be a simple objective based game such as moving an entity to complete a goal. The agent should be able to learn how to complete its goal and adapt over time to perfect it's method. It will be a 2D game ran in a 3D environment where the agent adapts to oncoming objects and jumps over them, the agent learns through reinforcement such as a reward system, while a score counter keeps check of how many objects have been avoided (jumped over).

1.2 Objectives

1. Main Objectives

    1.1. Learn how to implement the ML-Agents framework into the unity game engine

    1.2. Develop standalone game, with main functionality

    1.3. Implement Artificial intelligence into the game so that it may be autonomous

    1.4. Train the model so that it can learn how to play the game

    1.5. Review results

    1.6. Discuss future optimizations

2. Optional Objectives

2.1. Use other libraries for Machine Learning

2.2. Use Deep learning and GPU acceleration to enhance performance, measuring linear time against parallel time

2.3. Add additional functionality to the game such as more movement or shooting or different types of objects

2 Literature Review and Background

This chapter will relay foundational information on Machine Learning techniques which can be applied to game engines and a summary of it's background.

2.1 Background Research

AI in the gaming industry has been an essential facet of game production and consumer engagement since 1970, with the creation of the arcade game 'Computer Space' by Atari. The purpose of AI in the arcade era was to give games replay value, at a time where every play-through required a payment. Arcade games of that era ran on simple scripts and randomized behaviors to avoid consumers learning a pattern that would allow them to win consistently. It's this very challenge that created a learning curve and an obsession with reaching the game's final stage. This early kind of 'AI' appeared in the form of hard-coded stored scripts and patterns, which meant that there was no real machine-learning, only prompts based on actions and script-triggering decisions by the player. Bohannon (2015).

Games like Pong and Pac-man took it a step further and created the first instance of a computer opponent with artificial intelligence. The computer opponents in these games were designed to make decisions based on the the the player's actions. Despite the simplicity of the AI, this gave people the illusion that the game had a mind of its own, and was thinking on its own accord, which was an essential milestone for AI in the industry. Robles and Lucas (2009).

With the introduction of home-computers and PCs, modern processing power has improved significantly since the Arcade era. Therefore the use of more complex and unpredictable AI in games became industry standard. Consumers started paying full price for an unlimited amount of play-throughs, and expected longevity in exchange for their money. This pressure pushed game developers to create more engaging and challenging genres, and amongst them was Real-time strategy; this was the pivot point in the industry and introduced a highly precise AI that factor in multiple scenarios and decisions before responding the a player's prompt. Mohri and Talwalkar (2012).

The biggest improvement to gaming AI in recent years is the machine's ability to learn and adapt to the scenarios it is in. And as games are advancing constantly, AI is constantly being challenged into new heights in regards to machine learning, and its role in providing a user with the most realistic experience possible. The key to this is balance; there are fundamental

advantages that AI has over humans, and that is the ability to think as fast as it is programmed to think, and to multi-task multiple commands and scripts simultaneously, while also having a slowly growing understanding of contextual gameplay. Context and gameplay go in hand in hand when it comes to AI and machine learning, the more varied the gameplay is, the more complex the context, which in turn requires a more advanced AI to accomplish the task of providing the player with a realistic engaging experience. Narula (2019)

Machine learning algorithms have been vastly used in the software development industry, one of the most successful subcategories is the computer vision sector. When applying machine learning concepts to games, we can take some of the methods used from computer vision and apply them to machine learning in games. Intelligence can be perceived by humans when an entity shows it has a set of cognitive processes which work together or independently (Copeland, J. (2000)):

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

AI applications nowadays contain some of these processes, such as language understanding and the ability to learn, we will take some of these concepts and apply them to the game model.

2.2 Training Methods

Four training methods present in machine learning in particular are important to investigate for this application; reinforcement learning, imitation learning, supervised learning and unsupervised learning. This research project will incorporate a mix of Neural networks, supervised learning, Reinforcement learning and Deep learning.

1. **Reinforcement learning** derives from real world examples of how mammals perceive rewards and punishments. The main concept derives from psychologists and researchers whom have researched conditioning and reward systems, such as Skinner (1937) who developed the Operand Conditioning theory in 1937, and Pavlov (1927) who researched conditioning on dogs. The main aspect of the theory in psychology is using stimuli as rewards/punishments to condition mammals to lean to do what we want them to do.

The mechanism of this system in computer science thrives on a reward/punishment system. The state machine passively interacts with the environment, the machine collects a reward or punishment for each of its actions. The goal of this system is to increase its reward or decrease the risk over a sequence of actions and iterations with the environment, the more point's the machine obtains, the closer it is to it's goal, it can also lose points by doing incorrect actions. As depicted in Figure 1, reinforcement learning algorithms keep learning from experience of the environment until they explore all possible states.

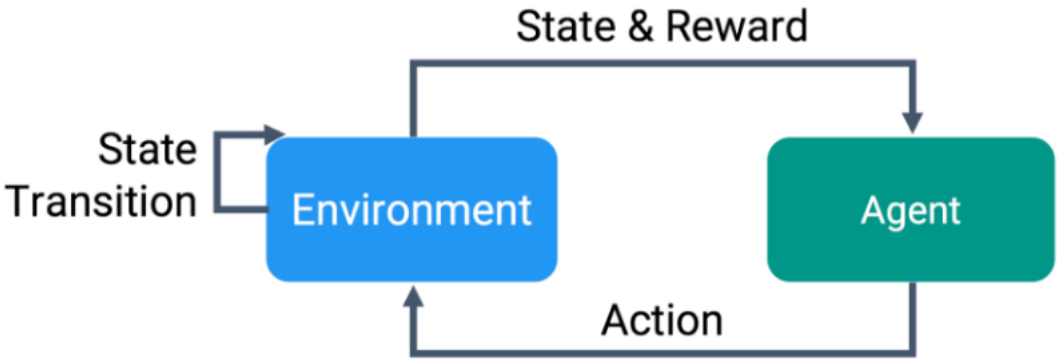


Figure 1: Reinforcement learning and rewards

A couple algorithms use reinforcement learning such as:

1.1. Q - Learning

1.2. Temporal Difference

1.3. Deep Adversarial Networks

2. **Imitation learning** is a method where the program learns from demonstrations, this method is used when the reward function is difficult to depict (Arthur Juliani (2018)).

3. **Supervised learning**; is where the machine learning model is given training data. This set of data contains scenarios, related with outcomes, these are labels for the scenarios. Supervised learning depends heavily on statistical maths, using algorithms such as linear regression, nearest neighbor and decision trees to predict certain outcomes. Supervised learning uses these algorithms paired with the training data to accurately predict outcomes when given a new data set or scenario (Mohri and Talwalkar (2012)).

4. **Unsupervised** Learning is a method where you input the data but only the scenarios are inputted but not the labels. The machine builds a model to predict the output of new data. This can be done by comparing and grouping similar training data which is called clustering Mohri and Talwalkar (2012). Some of the common algorithms that use unsupervised learning are:

    4.1. Association rules

    4.2. K-means clustering

2.3 ML-Agents Framework in Unity

In the ML Agents framework Agents are actors and the behavior which is linked to them determines how they act, an agent requires a behavior in order to function. The agent is responsible for collecting observations, executing actions and assigning rewards. The behavior entity of the framework is responsible for obtaining the observations and rewards and is responsible for determining which action to execute. As depicted in figure 2.

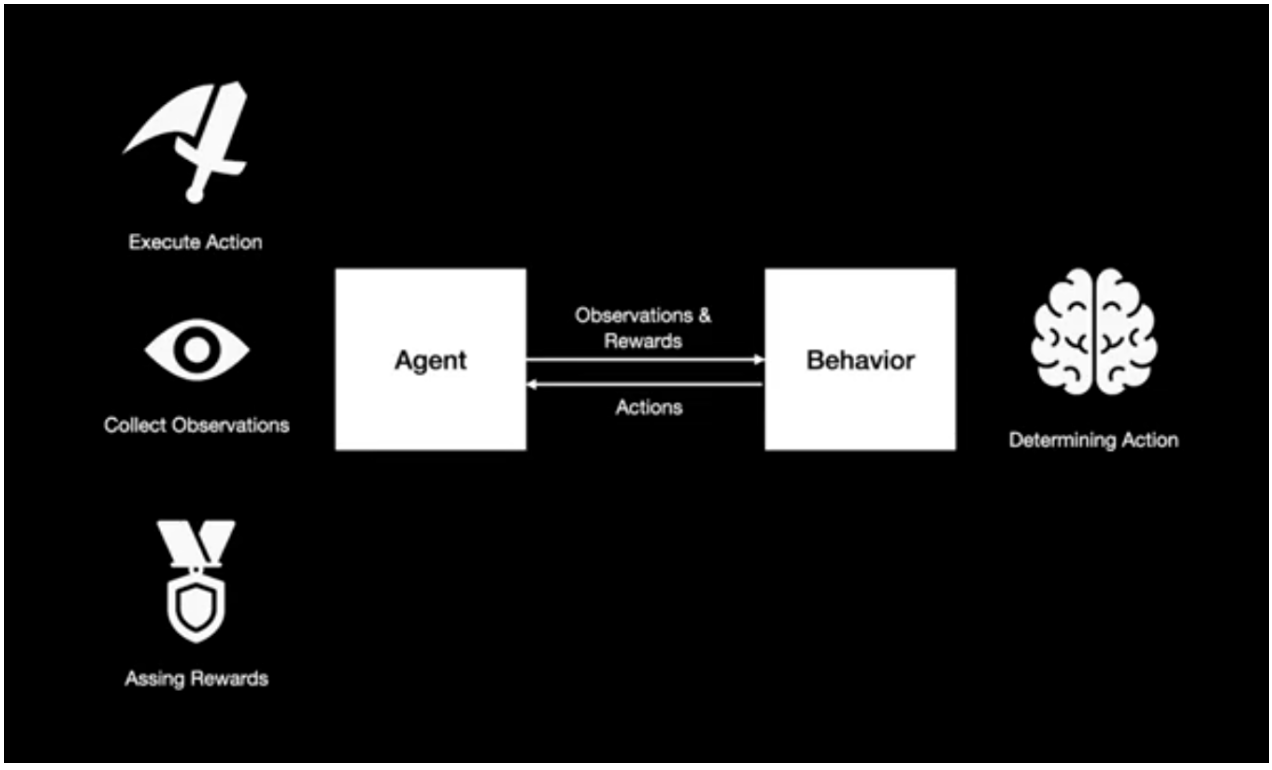


Figure 2: Agents linked to behavior

ML-Agents uses ray casting to observe the environment and store data in vector observation arrays. These are arrays which contains floats of vectors in the program which can be sent to the

neural network. Ray casting in Unity is a powerful and common tool to observe the environment. The behavior parameters in the framework consist of vector observations, discrete or continuous data and model behavior types. Model behavior types consist of three main types.

1. **Heuristic:** This is a classic implementation model, where the programmer decides how they want the intelligent agent to work and develop a hardcoded script to do certain actions given a specific state or response to a change in external stimuli. However this is not machine learning per say as it lacks versatility and can't change.

2. **Learning:** When the AI is currently trained using machine learning, during training a neural network gets generated, and in order to use the learned model and apply it to the system the last model used is inference.

3. **Inference:** The learned model is applied to the system but not changed.

2.3.1 Agent

The foundations of the agent script depend on three main aspects, the initialize function, the collect observations function, on action received fucntion and on episode begin function. When creating a custom agent you can inherit the initial agent superclass and can inherit these functions.

1. **Initialize method:** this is called when the game object gets enabled, this happens between the awake(); and start(); function in unity. In the base class the agent connects to the behavior and the Academy.

2. **Academy:** is an important aspect of this framework. It is a globally accessible singleton for all agents. The academy controls the training process for all agents, and is in charge with syncing them. Global parameters can be set in the academy and are accessible by all agents.

3. **Collect Observations:** Everything important the AI needs to make a useful decision is collected in this method.

4. **On Action Received:** This method allocates actions for the AI to execute, whether it be moving or shooting or re-spawning etc.

5. **On Episode Begin:** An episode lasts until the objective has been completed or failed.

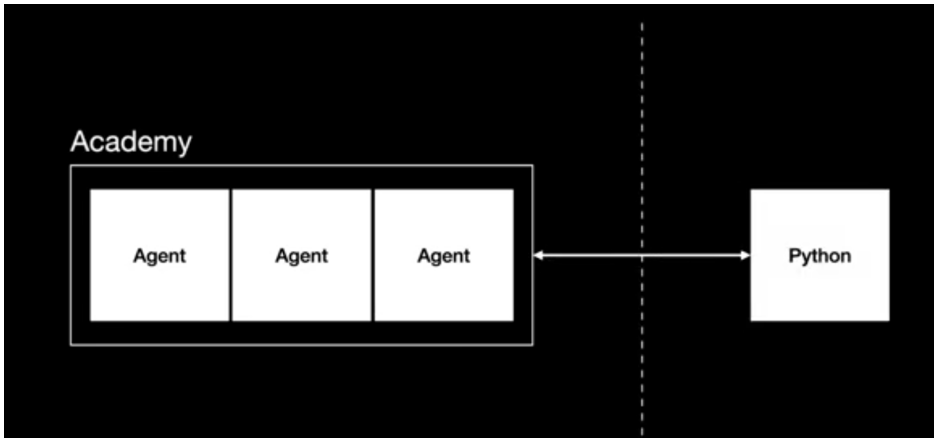


Figure 3: Academy linked to multiple agents

2.3.2 Brain

The Brain and the agent are linked together, the brain decides the agent's actions. The brain in this framework takes a stack of vector observations, these are vectors containing floats about the environment and inputs it in the neural net to calculate and make a final decision. The brain makes it's decisions via four different entities, it has a separate mode for each of these mediums, These modes are Player, Heuristic, External and Internal.

1. **Player:** The player controls the decisions made by the agent by reacting to the player's inputs. We will not be using this function as it requires no machine learning.

2. **Heuristic** The decisions are hardcoded by the developer.

3. **External:** Decisions are executed via an external communicator using machine learning libraries such as TensorFlow

4. **Internal** The Agent's actions are decided using a trained model. For this implementation, the model gets generated after the training process, by TensorFlow, then it is embedded into the brain using TensorFlowSharp.

2.4 Critical Thinking

Despite the benefits of machine learning in gaming, the goal isn't to achieve the most intelligent AI when it comes to developing a game, because that is all subjective to genre and story. The goal is to create a user experience. So in essence, while there can be infinite room for progress and growth for AI in the gaming industry, there doesn't necessarily have to be any further

research, as for the current standard of games, they only require a number of heuristic functions to achieve that goal. This is where the distinction between academic AI and gaming AI is made; Academic AI is a boundless search for full automation of machine learning and adaptation in the hopes to one day replicate the human conscience, and break as many boundaries as possible. Gold (2005)

Not much research has been conducted in the field of improving performance for Artificial Intelligence on hand held devices, in order to enhance performance, lower level programming such as embedded and processor programming such as Assembly would be useful. One of the main ways to improve computation time for machine learning methods is using GPU acceleration and parallel processing. Much research has been conducted in this area, such as improving clustering performance and k-means algorithms Ren Wu (2009). Another study from Harvard by Lu et al. (2010), used high resolution images from the International Space Station and split the data into multiple segmentations and ran them on the GPU, this study shows the power and high performance capabilities of graphics cards. However, many handheld devices have limited GPU's or use on-board/integrated GPU's, this eliminates the capabilities of running massive neural networks on the GPU for handheld devices, this is because aftermarket GPU cards (for example NVIDIA's Tesla), contain hundreds of GPU cores which can all run in parallel and simultaneously execute instructions. The other issue is aftermarket graphics cards have large global and shared memory, which allows the GPU to process data in larger orders of magnitude than when just using internal memory on hand held devices.

One study Lieber (2019), used virtual machines to run android in a virtual environment and machine reinforcement learning to train the AI. This was a true virtual environment as it emulates the exact ARM instruction set for android processors. Using this virtual environment, it was possible to use CPU acceleration for the android environment by using the CPU hardware. The study used visual preceptors and visual observations to input into the neural net and have the autonomous agents react accordingly. After 25 million steps and running 12 emulators simultaneously the total accumulated reward for the Flappy Bird game was 150. The agent successfully played the game without dying for 30 minuets.

To summarize, not much research has been investigated to train AI on handheld devices, in order to train advanced autonomous agents, it is crucial to have the right hardware, one way to implement this could be to train the AI on an external GPU and port it to handheld devices.

3 Requirements Analysis

This paper is intended for research purposes, the aim is to develop an autonomous agent in the unity game engine using the ML-Agents framework developed by Unity Technologies. There are no human interactions with the program, it will be an agent which learns to play the game itself through machine learning algorithms using learning techniques and neural networks to train the machine.

However, the program itself must be developed and the main functionality and foundations must be defined. Figure 4 contains the list of functional requirements crucial to the development of this game.

ID	Requirements	
FR.1	The learning agent must be autonomous	Must Have
FR.2	The agent must be able to learn	Must Have
FR.3	The agent must be able to perform basic movement	Must Have
FR.4	The agent must jump	Must Have
FR.5	The game must allow for basic movement	Must Have
FR.6	The game must have a score for amount of jumps	Must Have
FR.7	Multiple instances must be ran simultaneously	Must Have
FR.8	The agent should be able to predict incoming objects	Should Have
FR.9	The agent should be able to lose the game/die	Should Have
FR.10	The agent could be able to adapt its own play style	Could Have
FR.11	The game could have sound effects and music	Could Have

Figure 4: Functional Requirements

4 Design and Implementation

This section will briefly overview the different solutions for an out of core model to partition and transfer vectors for computation on the GPU, it will describe how to do it. Seven separate programs were created and analysed to compute vector operations. The initial concept will first be covered, then the actual code and theory will be explained.

4.1 Design

Before creating the programs, it is important to have the correct layout and design for the base functionality of the implementation. The programs provided will be written in C# for Unity. The game is a 2D game ran in a 3D environment. There will be one humanoid model (the player/agent) and one car model which heads towards the human and he has to jump to avoid the car. The game models used are open source and available on the unity asset store for free.

The character model is depicted in figure 5. The layout of the game whilst in play is depicted in figure 6

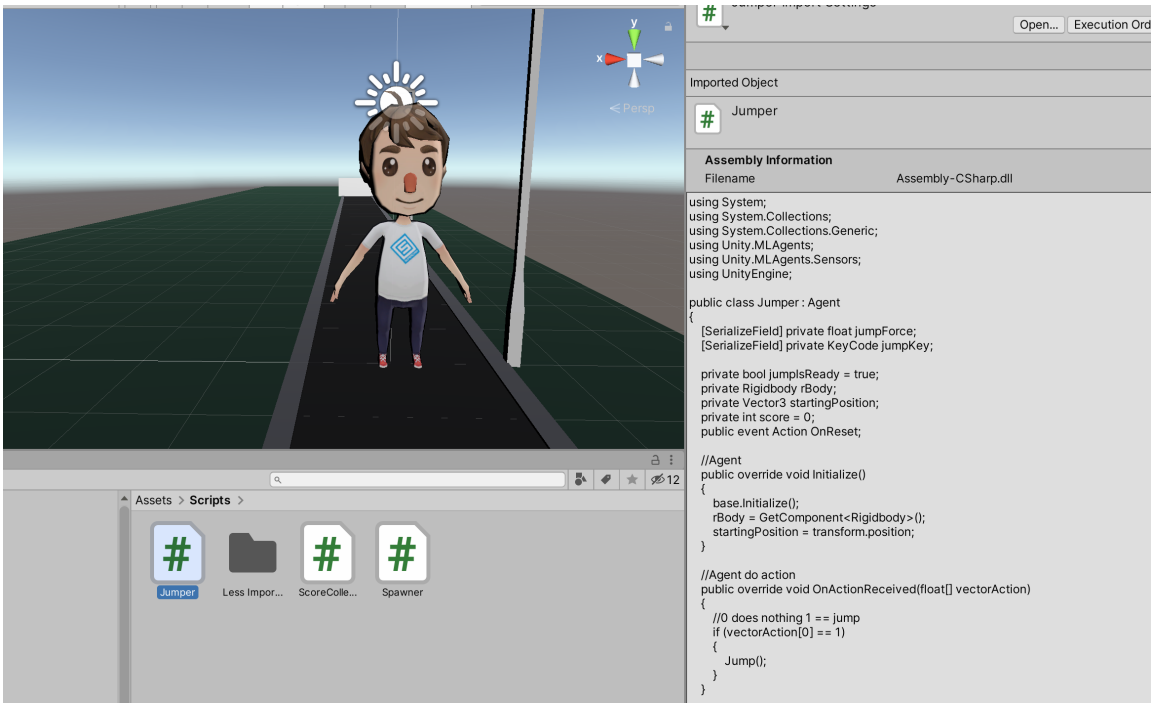


Figure 5: Character model with scripts and initial starting position.

The format and stages of the implementation are listed as follows.

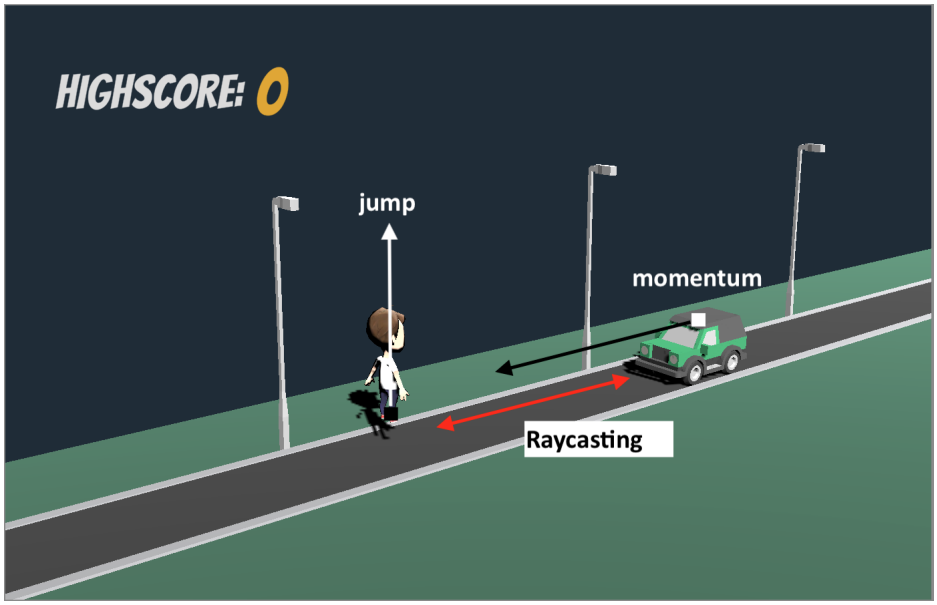


Figure 6: How the game appears when played. The object moves forward the agent jumps using ray casting to determine the distance between itself and the object

1. Create base game
    1.1. Character script contains:
        i. Action response to input (i.e. space key to jump)
        ii. Basic Collision detection
        iii. Reset game parameters if collision
    1.2. Game spawner: spawns enemies
    1.3. Enemy Script: Move towards agent with collision detection
2. ML-Agent framework
    2.1. Change initial player script to inherit from ML Agent superclass
    2.2. Train the agent with multiple instances of the agent running simultaneously
    2.3. Loop:
        i. Train until the score can be reached higher than 8

4.2 Implementation

This implementations section mainly covers the changes made to the initial file to a child of the Agent superclass. This section discusses the code required to move into overridden functions

from the superclass. Firstly the ML-Agent framework must be linked to Unity. The Git Repo for ML-Agents was cloned and the assets and scripts were imported into the dissertation project. (Unity Technologies (2018)). Then the foundations of the game were created, the environment and surroundings using free assets. The scripts were created for spawning enemies and collision, a score collector script which accumulates the score and displays it and the jumper script linked to the player model. The specific details of which for each script will not be covered as this paper looks at machine learning in games, therefore the implementation will cover the Machine Learning Framework. Having mentioned this, there should be a fundamental understanding of how physics in games work and how scripts are attached to entities in Unity, these tutorials can help Unity (2020).

Free assets were accessed via unity's asset store and added to the project. Scripts are then attached to the models.

After creating the main scripts, the brain had to be attached to the entity which we want the agent to control, as seen in figure 7. This controller takes in an input of vector observations, we don't need a stack of vectors as we are only observing one thing and performing one action therefore the stack vector is 1. Firstly we will use CPU sequential execution and not GPU acceleration and we will set the behavior type to default. Once the Agent is trained then we can input the trained model into the brain and use it to play the game, we are using reinforcement learning to teach the brain then supervised learning to apply the training model to the game.

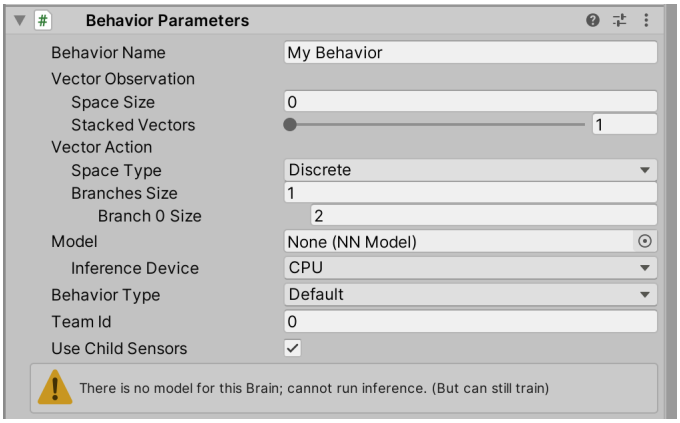


Figure 7: Brain controller for information inputting in the neural net

The next step was to initialize ray casting on the model which we want the agent to move, the white lines for ray casting can be seen in figure 8.

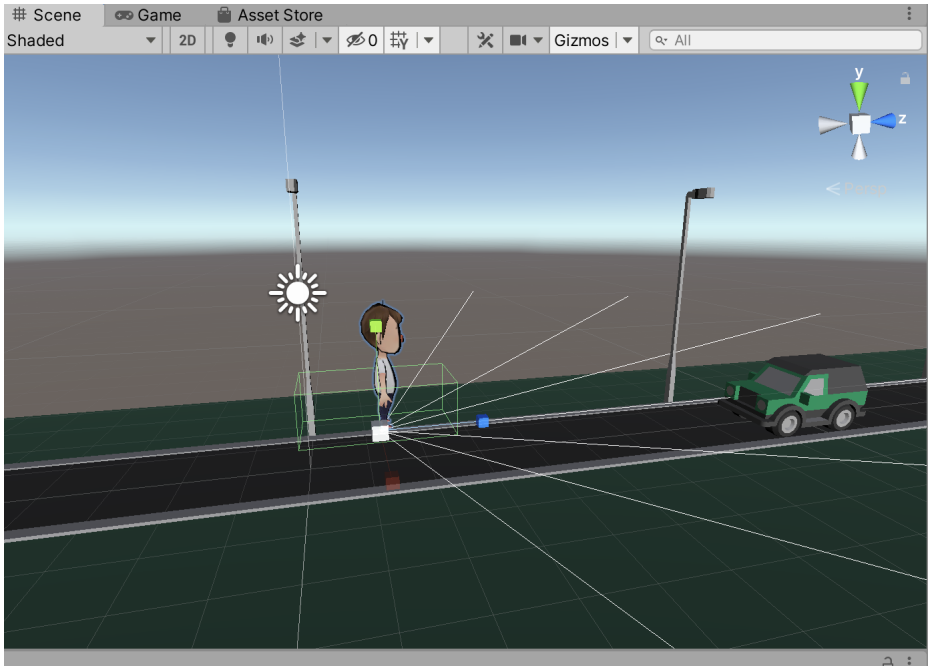


Figure 8: Ray casting activated for Agent

The initial change which must be done in the code is inheriting from the ML agents superclass. Then we can move the code from the *Awake*(); function to the *Initialize*(); function.

```
//Jumper inherits from superclass Agent (ML-Agent framework)
public class Jumper : Agent
{
      private bool jumpIsReady = true;
      private Rigidbody rBody;
      private Vector3 startingPosition;
      private int score = 0;
      public event Action OnReset;

      //Agent
      public override void Initialize()
       {
            base.Initialize();
            rBody = GetComponent<Rigidbody>();
            startingPosition = transform.position;
      }
```

As mentioned before, heuristic generally refers to hardcoding an entity, for testing purposes we move our check for user input into the heuristic function. To check for input, the initial code was in the unity global *Update*(); function. This function was used to check for user input, since we are making our agent autonomous we add this get input function to the heuristic function which we override from the superclass.

```
//Agent HUERISTIC.
    public override void Heuristic(float[] actionsOut)
    {
        if (Input.GetKey(jumpKey))
            Jump();
    }
```

Action handling in ML-Agents is monitored and controlled by the *ActionRecieved*(); function. The vector containing actions is in *ActionsOut*[] anything contained in ActionsOut will be sent and handled by the *OnActionRecieved*(); function

```
    public override void OnActionReceived(float[] vectorAction)
    {
        if (vectorAction[0] == 1)
        {
            Jump();
        }
    }
```

As shown above the *OnActionReceived* function checks the vector array and if it contains an action in the first slot then it will call the jump function.

The next step is to add rewards to the agent, the initial idea is that, when the object collides with the agent/player then the agent decrements the reward by -1, when the agent jumps over the object, it collides with a hidden object, this hidden collision increments the reward by 0.1.

```
        private void OnTriggerEnter(Collider collision)
            {
                 if (collidedObj.gameObject.Tag("score"))
                    {
```

```
                AddReward(0.1f); //ML Agent
                score++;
            }
        }
```

The script must check, if there is collision with the player and the street then it can perform a jump, else it is in the air, and else if there is a collision with an object then it will decrement the reward. If there is a collision, then there will be a de incrementation in the reward and the game resets because the agent lost, and the end episode function must be called to end the episode.

```
private void OnCollision(Collision collidedObj)
 {
    if (collidedObj.gameObject.Tag("Street"))
        jumpIsReady = true;

    else if (collidedObj.gameObject.Tag("Enemy"))
    {
        AddReward(-1.0f); // decrement the reward, punish the agent
        Reset(); //reset the game
        EndEpisode(); //End the episode (ML AGENT FRAMEWORK CALL)
    }
 }
```

Lastly to train the AI, we use python's integration with ML-Agents, the command to execute the reinforcement training model is as follows:

```
mlagents-learn training.yaml --run-id="dissertation"
```

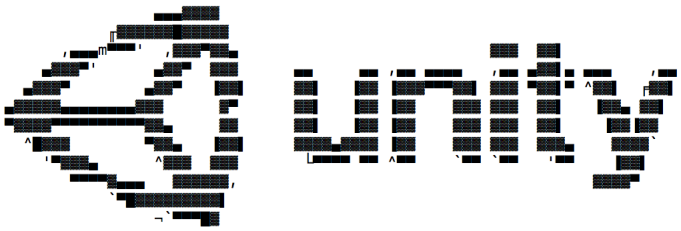
After this, the terminal prompts to press the "play" button in the unity game engine to begin training the AI as shown in figure 9

Figure 10 shows 12 AI's being trained.

Then I built the app using

```
mlagents-learn train2.yaml --run-id=JumperFinal1
    --env=/Users/rashid/Desktop/rash/1MASTERS/Dissertation/Workspace/A.I.-Jumping-Cars-ML-Agents-E
```

```
WARNING:tensorflow:From /usr/local/lib/python3.7/site-packages/tensorflow/python/compat/v2_compat.py:96: disable_resource_variables (from tensorflow.p
scope) is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
```



```
 Version information:
  ml-agents: 0.18.0,
  ml-agents-envs: 0.18.0,
  Communicator API: 1.0.0,
  TensorFlow: 2.3.0
2020-08-02 18:55:25 INFO [environment.py:199] Listening on port 5004. Start training by pressing the Play button in the Unity Editor.
```

Figure 9: Running the training model

```
--time-scale=10 --quality-level=0 --width=512 --height=512
```

This image is the build running and training

it exported the Neural Network

Then I imported it into the brain and ran the program

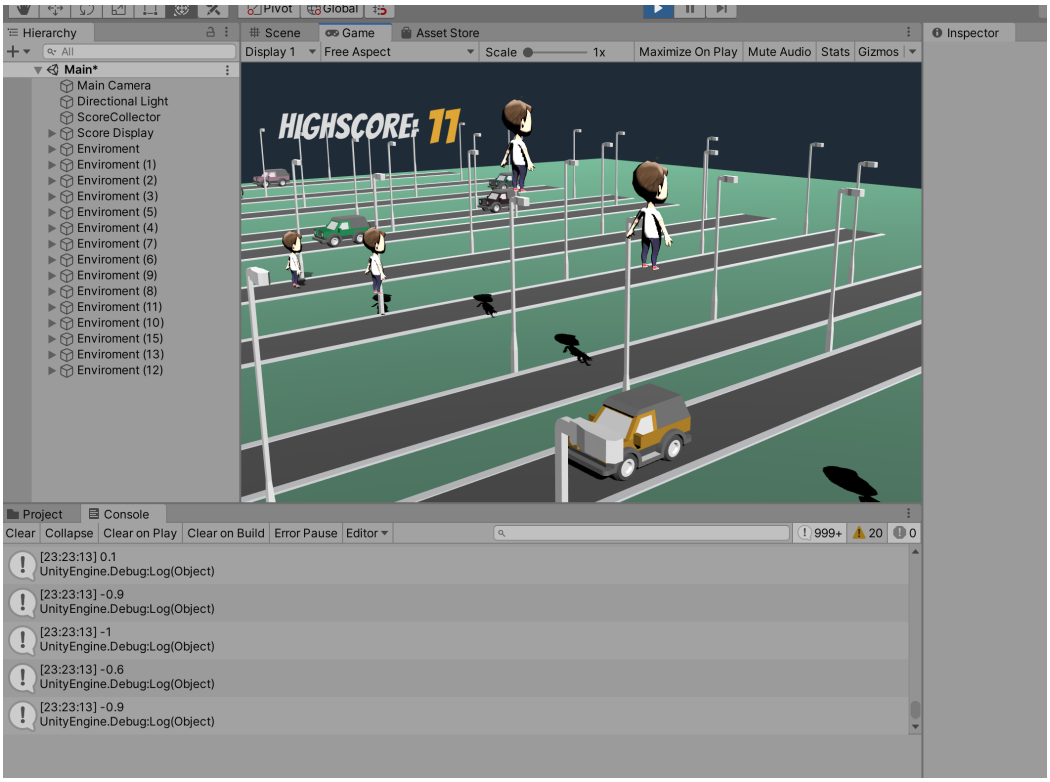


Figure 10: Multiple AI's being trained

```
trainer_type:   ppo
hyperparameters:
  batch_size:   1024
  buffer_size:  10240
  learning_rate:        0.0003
  beta: 0.005
  epsilon:      0.2
  lambd:        0.95
  num_epoch:    3
  learning_rate_schedule:       linear
network_settings:
  normalize:    False
  hidden_units: 128
  num_layers:   2
  vis_encode_type:      simple
  memory:       None
reward_signals:
  extrinsic:
    gamma:      0.99
    strength:   1.0
init_path:      None
keep_checkpoints:       5
checkpoint_interval:    500000
max_steps:      500000
time_horizon:   64
summary_freq:   50000
threaded:       True
self_play:      None
behavioral_cloning:     None
```

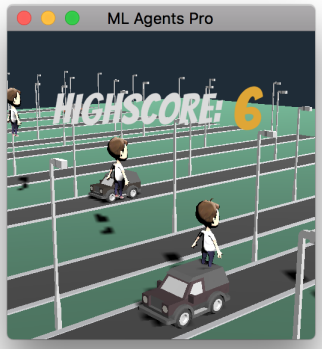


Figure 11: Build Application with HYPER PARAMETERS

```
IN: 'vector_observation': [-1, 1, 1, 3] => 'policy/main_graph_0/hidden_0/BiasAdd'
IN: 'action_masks': [-1, 1, 1, 2] => 'policy_1/strided_slice'
OUT: 'policy_1/concat_2/concat', 'action'
DONE: wrote results/JumperFinal1/My Behavior.nn file.
2020-08-03 20:23:26 INFO [model_serialization.py:83] Exported results/JumperFinal1/My Behavior.nn file
2020-08-03 20:23:26 INFO [environment.py:418] Environment shut down with return code 0.
```

Figure 12: Exporting the Neural Net

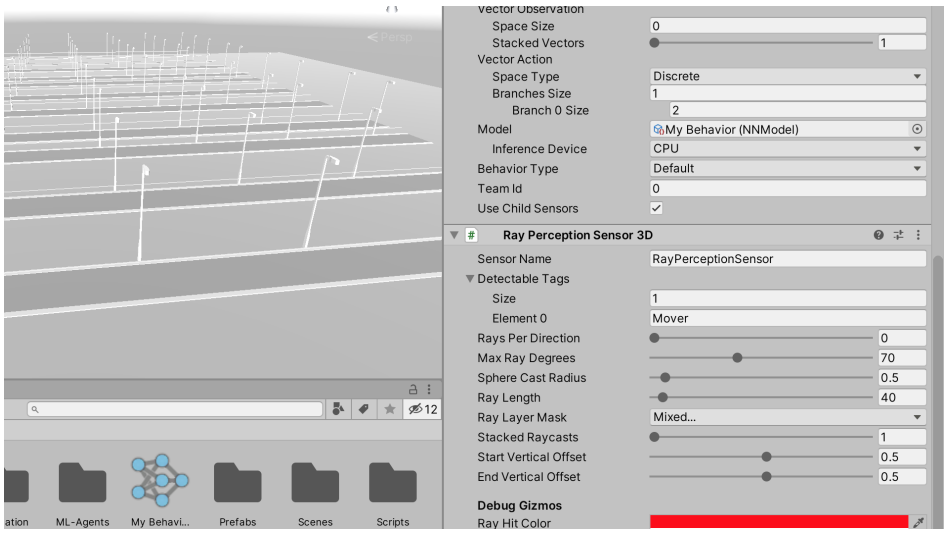


Figure 13: imported the brain into the program

5 Research Methodology

This chapter will describe the experimental design of the project, discussing first the main goal, the benchmarks and environment which it will be conducted in, the stages on how I will implement the project in relation to the objectives the paper sets out to achieve, lastly I will discuss the evaluation strategy in place.

This project conducts exploratory research in developing ML agents in the unity engine. The game will be a simple game for the AI to react to incoming cars and jump over

5.1 Testing Environment and Tools

These are the tools used for this project:

- Unity Engine
- Unity ML - Agents Framework
- Python
- C#
- NVIDIA CUDA
- Tensorflow

5.2 Benchmarks

Note for marker: Due to Covid-19, I was unable to attain any other devices to run my application on, therefore I used the MacBook Pro, the benchmark details are mentioned in this section.

I will develop two to three benchmark applications to test this system. This application will contain multiple instances of the agent, up to 14. The system is currently ran on the hardware specifications referenced in figure 14

5.3 Stages of Evaluation and Implementation

paragraph This section discusses developing the initial game, adding an autonomous machine learning agent and testing the environment.

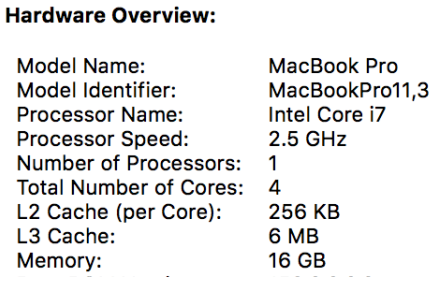


Figure 14: Running the training model on hardware

- Stage 1: Learning the unity framework and creating/collecting models for the game. Researching ML Agents and developing the foundations of the game functionality, such as jumping and score. As per objectives 1.1 and 1.2.
- Stage 2: Implement the ML-Agents framework into the base game to make the agent jump and react to oncoming objects, as per objective 1.3
- Stage 3: Train the model using reinforcement learning and ray casting to observe the environment and react accordingly, as per objective 1.4
- Stage 4: Train the model using multiple agents
- Stage 5 (optional): Run the training on GPU acceleration
- Stage 6: Record and evaluate results using TensorBoard

5.4 Evaluation Strategy

Rigorous and thorough tests based on benchmarks of the program will be conducted. To record these results I will use Tensor Board to analyse the cumulative reward gain over time.

1. Build final program and run the training process in the built environment for 1 agent.
2. Build final program and run the training process in the built environment for 12 agents.
3. Use tensorboard to evaluate total steps and concurrent reward over time
4. (Optional) Time execution from start to finish for GPU program
5. Compare findings.

6 Results and Evaluation

This section is a post-brief on the different programs, using the results collected to highlight which methods are optimal for certain situations.

All programs were observed on the MacBook Pro setups as previously mentioned, the results for the programs on the MacBook architecture will be revealed

Due to COVID-19, I was unable to attain different hardware devices therefore unable to present more results. The next few tables and graphs are results collected on the MacBook the results collected on this system.

Figure 15: Running the training model with multiple AI compared with just 1 AI

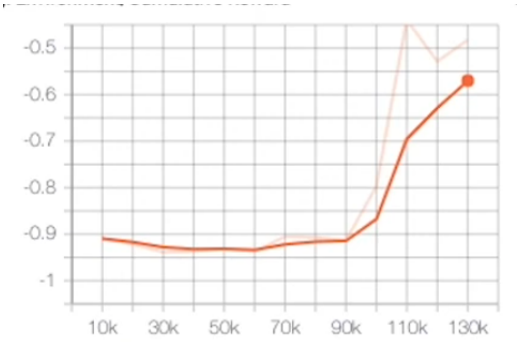


Figure 16: Running the training model with 1 AI

Interestingly enough, when running the program using the brain, the AI found a glitch, where just on collision with the oncoming car it would jump and fly very high into the sky thus having more air time.

6.1 Evaluation

Training with multiple instances of ML-Agents prove to be more efficient over time, increasing the hardware capabilities on handheld devices would be able to run ML Agents better. However, one could use a Virtual Machine using a different hardware environment but the same game in order to train the AI. Thus being able to make the Agent learn how to play the game and then once the Agent has learned, import it into the brain/NeuralNet and let the Agent play on the handheld device.

7 Problems Faced & Future Work

7.1 Problems Faced & Future Work

In the risk management section it had occurred that if an event out my control were to happen it would have an impact on the dissertation, COVID-19 set back my health and my ability to obtain any other hardware devices, thus was unable to provide test results on multiple architectures.

Another bug which continuously occurred randomly would be during training, the program would crash around 10 minuets, this set my project back as the neural network training was breaking after 10 minuets.

For future work it would be advantageous to implement the techniques used here towards developing an autonomous agent which can be ran on smaller devices.

8 Risk Management

This section will cover potential risks, the possibility of occurrence and potential management of said risks in tabular format.

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

Figure 17: Risk Analysis

9 Professional, Legal, Ethical and Social Issues

This chapter discusses legal and ethical issues related to my work.

9.1 Professional

As a computer scientist professional and student of Heriot Watt University, it is my responsibility to ensure that my dissertation is applied in a professional procedure. As the author I accept full responsibility for this project and the consequences resulting of. It is my job as a representative of Heriot Watt University, to make sure that no harm comes to its reputation as a result of unacceptable stands of professionalism. In regard to this, attention must be brought to the Heriot Watt University Code of Good Practice in Research, this is taken into full account for my research project. The program (Msc Artificial Intelligence) which this project is being done under is accredited by the British Computer Society (BCS), full acknowledgement will be taken towards their Code Of Good Practice.

9.2 Legal

The open source framework ML Agents, used in this project developed by Unity Technologies is under Apache License 2.0, A permissive license whose main conditions require preservation of copyright and license notices. Contributors provide an express grant of patent rights. Licensed works, modifications, and larger works may be distributed under different terms and without source code. This license allows for commercial use, private use and distribution. This project adheres to the apache license, the code of conduct set by Unity Technologies and the copyright notices of Unity Technologies.

Instances of abusive, harassing, or otherwise unacceptable behavior will be reported by contacting the project team at ml-agents@unity3d.com, as stated in their code of conduct (`https://github.com/Unity-Technologies/ml-agents/blob/master/CODE_OF_CONDUCT.md#enforcement`).

9.3 Ethical and Social Issues

There are no apparent human or social interactions which take place in this study, data or consent of humans are not collected, thus no ethical or social matters are present.

References

Arthur Juliani, H. H. (2018), 'Machine Learning Agents - Unity ML Agents Workflow WEBINAR [3/10] Live 2018/4/4 ', `https://www.youtube.com/watch?v=32wtJZ3yRfw&list=PLX2vGYjWbI0R08eWQkO7nQkGiicHAX7IX&index=1`. Accessed: 06-06-2020.

Bohannon, J. (2015), 'Artificial intelligence bests humans at classic arcade games'. Accessed 03-08-2020.

URL: *https://www.sciencemag.org/news/2015/02/artificial-intelligence-bests-humans-classic-arcade-games*

Copeland, J. (2000), 'What is artificial intelligence?', `http://www.alanturing.net/turing_archive/pages/reference\%20articles/what\%20is\%20ai.html`. Accessed: 06-06-2020.

Gold, A. (2005), A simple tree search method for playing ms. pac-man, pp. 249–255.

Lieber, O. (2019), 'Reinforcement learning for mobile games'.

URL: *https://towardsdatascience.com/reinforcement-learning-for-mobile-games-161a62926f7e*

Lu, Peter J. Oki, H. F. C. A., Chamitoff, G. E., Chiao, L., Fincke, E. M., Foale, C. M., Magnus, S. H., McArthur, W. S., Tani, D. M. and Whitson, P. A. (2010), *Orders-of-magnitude performance increases in GPU-accelerated correlation of images from the International Space Station*, Vol. 5, ISSN:1861-8219, DOI: 10.1007/s11554-009-0133-1, Published: Harvard.

Mohri, M., R. A. and Talwalkar, A. (2012), *Foundations of Machine Learning. Cambridge: The MIT Press*, Foundations of Machine Learning. Cambridge: The MIT Press.

Narula, G. (2019), 'Machine learning in gaming – building ais to conquer virtual worlds'.

URL: *https://emerj.com/ai-sector-overviews/machine-learning-in-gaming-building-ais-to-conquer-virtual-worlds*

Pavlov, I. (1927), *Conditioned reflexes: an investigation of the physiological activity of the cerebral cortex*, Oxford Univ. Press.

Ren Wu, Bin Zhang, M. H. (2009), *GPU Accelerated Large Scale Analytics*, HewlettPackard Development Company LP, HPL- 2009-38.

Robles, D. and Lucas, S. (2009), A simple tree search method for playing ms. pac-man, pp. 249–255.

Skinner, B. F. (1937), *Two Types of Conditioned Reflex: A Reply to Konorski and Miller*, The Journal of General Psychology. DOI: 10.1080/00221309.1937.9917951.

Unity, T. (2020), 'Unity Tutorials', `https://learn.unity.com/tutorials`. Accessed: 06-06-2020.

Unity Technologies (2018), 'Ml-agents', `https://github.com/Unity-Technologies/ml-agents`. Accessed: 06-06-2020.