MACHINE LEARNING IN VIDEO GAMES

FINAL PROJECT REPORT

DANIELE DI MODICA

Student no.   16819282
Supervisor   Stelios Kapetanakis
Second reader   Karl Cox

BSc Hons COMPUTER SCIENCE (GAMES) (CODE: J4EG027)
DEGREE OF BACHELOR OF SCIENCE

Abstract

Machine Learning in Video Games is an academic, one-year research and development project part of the undergraduate course under the module – CI301 Individual project at University of Brighton. It started in late September 2019 and has ended in late May/early June 2020.

The system, a 2D platformer game, is a small collection of levels, made in Unity, that use reinforcement learning to train an AI agent to play the game. The project's objective is to demonstrate that machine learning can be a viable, automated debugging tool for level design testing. As a matter of fact, the results of training the AI to complete the game prove that the AI is capable of discovering bugs and glitches, albeit in a random manner, within the levels.

Table of Contents

INTENTIONALLY BLANK PAGE

1. Introduction

First and foremost, the system attempts to demonstrate that by training an AI agent to play the game, machine learning can be used as a viable debugging tool, which may very well replace, to some extent, the role of game testers in the game industry as well as potentially increase the overall productivity during the development phase of a game.

Assume that an AI agent, which underwent a training process, is considered to be generic; it can complete any level within a game without having prior knowledge on how to solve it. The way it works is by feeding the agent with all the data it needs in order to be able to complete a level that features most variables and components that the game has to offer. The agent is rewarded for progressing and punished for digressing. As a result of this iterative process, the agent creates knowledge from past experiences. Once that knowledge has been acquired and the agent has learned the conditions to solve the problem, if then it faces unknowns during gameplay, it will still be able to complete the level (although it may take a few attempts to do so). The term generic refers to the theory that a machine, which possesses a hypothetical intelligence, has the capacity to understand/learn/perform any intellectual task that a human being can. Therefore, the agent can be used to perform tasks that are normally performed by a game tester.

Theoretically speaking, if the system implements an automated way to keep track of the status of the AI at any given point in time (by recording its inputs in the form of debug logs or any other visual aids, for example) the developers are therefore able to determine the nature of a bug as well as fix it. Needless to say, in its current state the system has only been able to achieve the most basic type of debugging which still requires human intervention by monitoring the agent playing the game (this can be done by recording the game session or simply by watching the agent play the game itself and check whether it finds bugs within the game). This is due to the fact that the system does not offer a way to store the information needed to debug the game when the system is left unsupervised.

Last but not least, this system was created as a way to study machine learning, which at the beginning of the project was a completely unknown subject. However, by undertaking this project, basic knowledge was gained on machine learning and more specifically on reinforcement learning, which is the training method used in the system.

The way the system was tested and the project deemed successful was thanks to the fact that the AI, whilst in training, was able to discover, level design bugs that would cause the agent to assume uncharacteristic behavioural patterns which at times would go against the rules of the game.

2. Methodology

The methodologies used in devising the system are mainly on the development side of the project. The management itself, due to poor management skills, underestimating the time it would take to design and develop the system, and private issues (aside from COVID-19) which slowed the progress down significantly for a few months, was never really implemented and the project was carried out as a result of day by day planning (see section 6.4 for more details).

Note: *Due to the nature of the project, no surveys were ever needed to be carried out.*

2.1. Project Management

The project never really had a proper management system implemented due to a series of circumstances external to the project. Except from the project plan made when the interim planning was submitted, no further planning was made.

2.2. Development Tools/Environments

The following sub-sections describe the tools and environments used in the project as well as provide explanation of choice for each.

2.2.1. Unity

Unity is a cross-platform game engine which consists of a physics and graphics engine and a live game preview. The latter allows you to view in real-time the changes made to the game during programming operations. Unity allows you to create 2D and 3D videogames or other interactive contents, such as architectural visualizations, 3D settings, shorts films, etc. Its engine allows you to code the game once and deploy it on different platforms. Unity uses a programming language called *UnistyScript* which in turn is made of two types of programming languages: JavaScript and C#.

Creating games is facilitated by many different tools that allow you to integrate the programming part of the game with the graphics requirements of the game; Unity is able to interact with Maya, Cinema 4D, Blender and other 3D modelling software. Another important feature is the presence of the Asset Store: a catalogue of 2D, 3D models, SKDs, templates, and tools available for purchase or download. In this way, even indie developers can optimize the workflow and speed up the development process in order to create a good product available on multiple platforms.

The reason why Unity was chosen to develop the system lies in the fact that the software has been used before and having familiarity with how it works, this resulted in increased productivity. This was done in order to avoid spending time learning how to use a new software, for example. Furthermore, Unity met all the criteria for designing and developing the system.
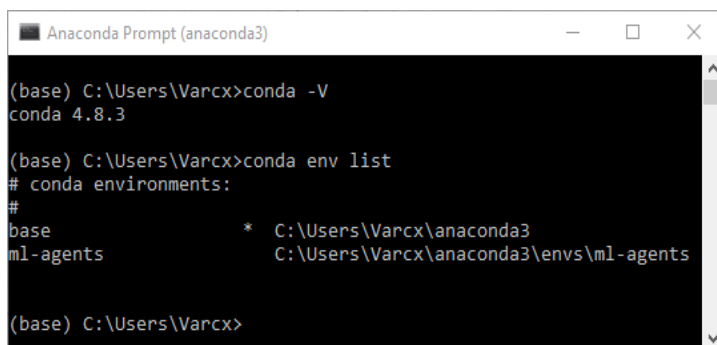
2.2.2. ML-Agents

ML-Agents is a GitHub repository of code that makes it possible to use neural networks and reinforcement learning for decision making in a game without any prior knowledge of reinforcement learning. The source code can be directly downloaded from GitHub without needing to use a version control software such as Git.

The reason why ML-Agents was used to implement machine learning was due to the fact that there was a need to find a way to create a system in Unity without needing to create it completely from scratch and most importantly without having prior knowledge of machine learning. So, even without having a strong foundation, it was still possible to build a system to prove the objective of this project.

2.2.3. Anaconda

This is a tool that manages Python installation and creates separate environments to easily train ML-Agents without needing to install Python. Essentially, it is a special command line that has specific Python libraries installed in different environments (Figure 1). This is useful because many times Python scripts take dependencies on specific versions of Python libraries.

Figure 1: The Anaconda command line showing the version installed on Windows as well as a list of environments.

It goes without saying that ML-Agents relies on Anaconda. And this was the reason why Anaconda was used.

2.2.4. GIMP

GIMP is a free cross-platform image editor and was the primary choice to cover the graphical aspect of the system. It was used to create the assets of one of the games (the system is comprised of two games – section 3 covers the specifications of each game) due to its intuitive and easy to use editor and the simple fact that the software has been used many times before, even outside this project, and therefore there was no need to use an alternative option.

2.3. Testing - TensorBoard

TensorBoard is a visualization toolkit which provides the tooling needed for machine learning experimentation. Tests involved hyperparameters tuning (see section 4 for more details) and training the agent (see sections 3.1.3, 3.2.2 and 3.2.4) while adjusting the hyperparameters to get better results.

TensorBoard is part of TensorFlow's tools, libraries and community resources. TensorFlow is an open source machine learning platform and when machine learning is involved (usually), so is TensorFlow. Not to mention that TensorBoard is integrated with ML-Agents which is the reason why the tool was used for testing and data analysis.

3. Product Description

The project architecture follows a simple Unity project folder structure in the form of a hierarchical tree structure as shown in the image below:
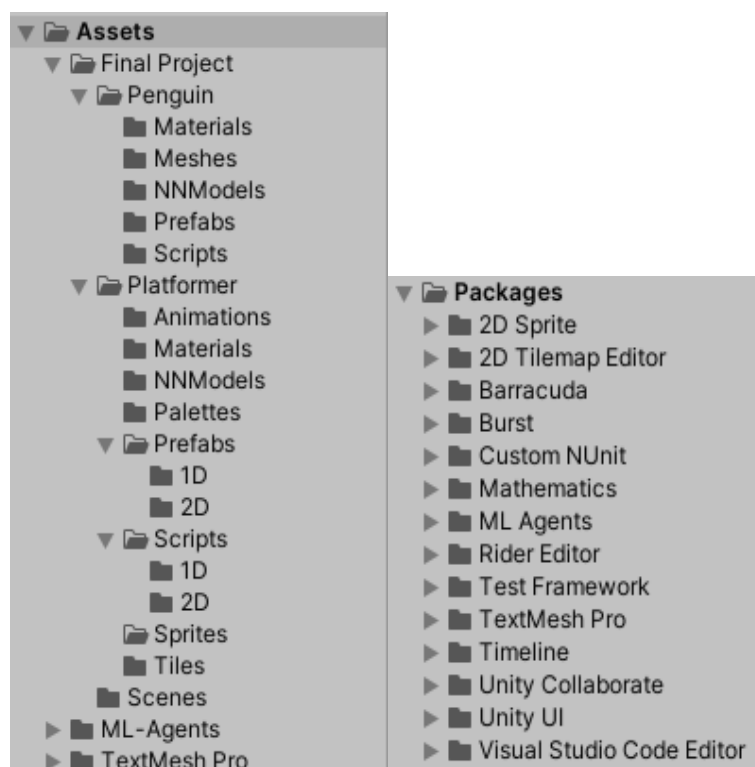

Figure 2: Unity folder structure of the entire project.

When a new project is created Unity automatically generates the *Assets* folder by default. This is the most important folder in a Unity project and contains all the game sources, including scripts, textures, sounds etc (Figure 2). The *Final Project* folder contains the core of the system comprised of two games, which are contained in the *Penguin* folder and *Platformer* folder respectively, and a *Scenes* folder

(Figure 3) containing the game levels of each game. The choice of having a shared folder for all the scenes of two different and independent games is due to the fact that they are part of the project which could be seen as one single game containing two mini-games.
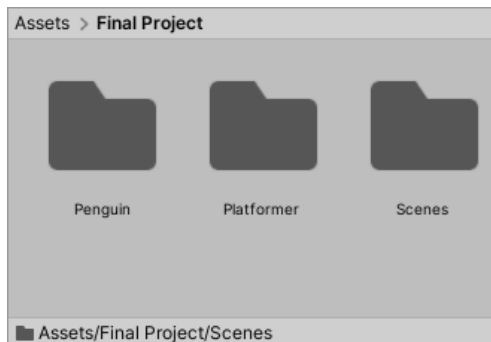

Figure 3: Scenes contains levels of both games.

As mentioned above, both games are independent of one another and can be considered to be two small projects within the bigger project. As a matter of fact, each game has a project structure of its own containing all the game assets and components (Figure 2).

The *Packages* folder, as the name itself suggests, contains all the packages used by Unity to implement additional functionalities such as *2D Sprite* that enables the use of *Unity Sprite Editor Window* to create and edit sprite asset properties like pivot, borders and physics shape).

3.1. Penguin

Penguin was made as a result of following a set of tutorials on reinforcement learning in Unity that can be found on Unity's website (ML-Agents: Penguins). They introduce Unity ML-Agents (see sections 2.2.2 & 4 for more details) and explain how to set up a ML-Agents environment and train an agent using machine Learning. Therefore, *Penguin* was used as an introductory, step-by-step guide which laid the groundwork to build *Platform,* whose structure is in fact based on Penguin's.
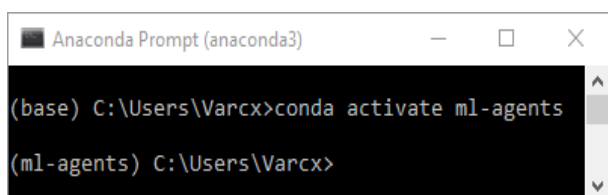
The game can be considered as a training ground for penguins to learn how to catch fish and bring them back to their babies. The game is comprised of one scene which contains several game areas, identical one another, where the penguins train. The use of multiple game areas makes for a more efficient training. Each game area is a *GameObject* containing all elements needed for one penguin to train. *GameObjects* are fundamental objects in Unity that act as containers for components which implement real functionality. A *GameObject* can represent anything from characters to props to sceneries.

The first step in making the game, as per instructions given by the tutorials, was to set up ML-Agents and Anaconda for Windows. It is worth mentioning that the tutorials on Unity's website use an older

version of ML-Agent (ML-Agents Beta 0.13.0) whereas both games discussed in this report use the 0.15.1 release. Migrating to 0.15.1 required consulting the [Migrating documentation page](#) which provides an explanation of what has changed and how to upgrade to newer versions.

3.1.1. Environment Setup

The Anaconda prompt was used to create a Python environment (see section 2.2.3) for ML-Agents training scripts. This step was needed in order to install all the packages required to create the environment. Once the environment setup was completed, the prompt was then used to install the content of the ML-Agents folder previously downloaded from GitHub. This step was necessary in order to use ML-Agents (Figure 4) in the training and inference phases of the game.

Figure 4: Command to activate the environment named ml-agents.

Finally, a Unity project was created and the ML-Agents code imported via the Package Manager. Prior to 0.15.1, to use ML-Agents in Unity, the directory downloaded from GitHub needed to be imported directly into the project in Unity for it to work.

3.1.2. Scene Setup

The next step was to create a scene that included a penguin, a baby penguin, several fishes, and a small enclosed area where the penguin could move around, collect fish and feed the baby (Figure 5).
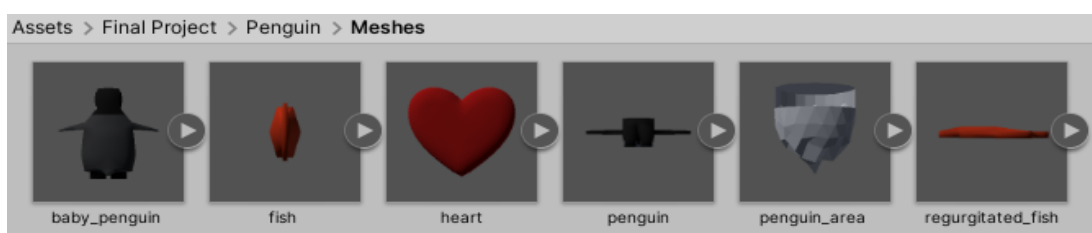

Figure 5: Imported meshes used in the game.

To do so, the aforementioned elements were downloaded from the website and imported in the scene. Each mesh is made of a 3D model and a number of materials. The scene was then assembled to make it look like as shown in the screenshot below (Figure 6):
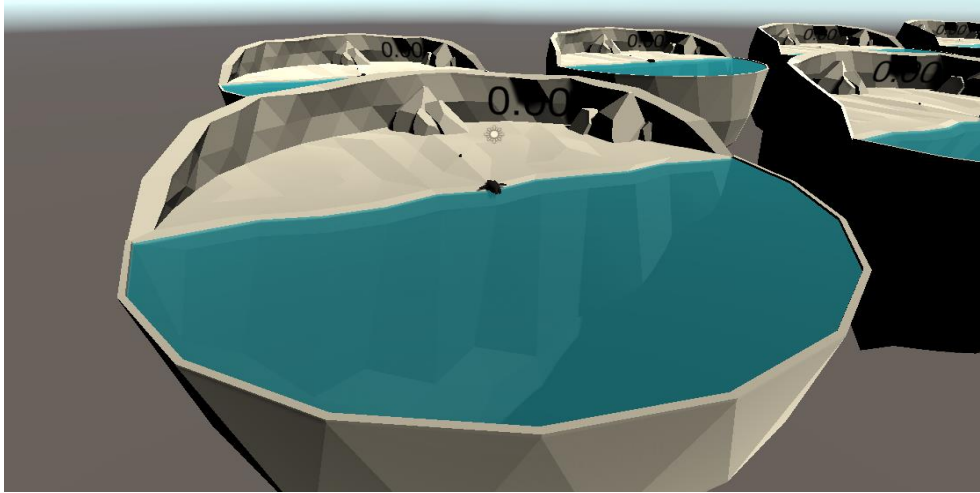
Figure 6: Multiple game areas.

A single game area (*PenguinArea*) manages a training area with one penguin, one baby and multiple fishes. It has the responsibilities of removing and spawning the fish and random placement of the penguins (Figure 7). It can be considered to be a parent object whereas all the elements inside are its children. Each object in the game, *PenguinArea* included, was then made into a Prefab so that they could be easily instantiated to create multiple game areas.
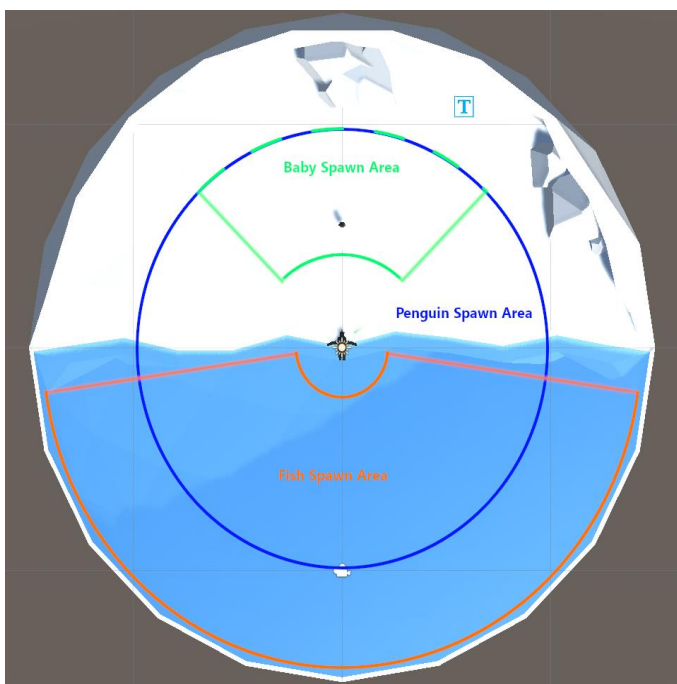

Figure 7: Illustration taken directly from Unity's website. It shows how the spawn areas are handled inside the PenguinArea.

A *Prefab* (Figure 8) is a *GameObject* whose configurations, with all its components, property values, and child *GameObjects* are saved as a reusable asset. The *Prefab Asset* acts as a template that can be instantiated in the scene as new *Prefabs*. This favours reusability thanks to the *Prefab* system that

keeps all the copies automatically in sync. Any changes made to a Prefab Asset are automatically reflected in the instances of that Prefab.
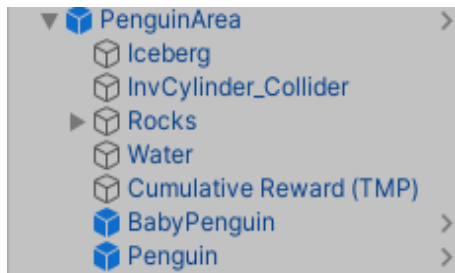

Figure 8: Hierarchy of PenguinArea showing Parent-Child relationship. Unity shows Prefabs in blue.
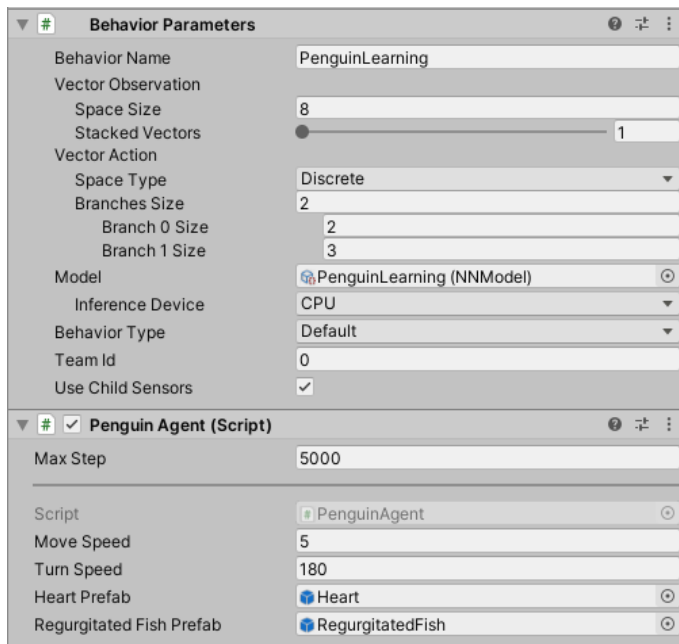
The *Cumulative Reward* is a text, placed in the scene, which displays the reward in a numerical format every time the agent performs an action with the goal of maximising the reward (see section 3.1.4).

Note: *By default, the agent is given a tiny negative reward to encourage action. A negative reward is normally given as way to punish the agent when it does something wrong. However, the documentation on the ML-Agents' website recommends against excessively punishing to shape the desired behaviour of an agent as this might negatively affect the outcome of the training; the agent might fail to learn any meaningful behaviour. In this particular case, a tiny negative reward forces the agent to complete the task as quickly as possible to get the maximum reward possible, which is the intent behind this specific training method.*

3.1.3 PenguinAgent

Once the scene had been set up, the next phase was to make the Penguin Prefab into an agent able to observe its environment and take actions using deep learning. This involved attaching a *Ridigbody* and *Collider* component to the Penguin GameObject as well as the script to manage the aforementioned actions. The script contains information so that an additional component is automatically added to the penguin once the script is attached to it; that is the *Behaviour Parameters* component.

This component manages the way the penguin learns. The most important parameter is the Behaviour Name (Figure 9) which needs to match the configuration files (further details are provided in the 3.1.4 section) needed to train the penguin. It is vital that the name in the Behaviour Parameters component matches the name in the configuration files in order for the agent to learn, otherwise the training would not work.

Figure 9: Behaviour parameters which define how the agent behaves.

The *Vector Observation Space Size* corresponds to a list of numerical observations on the environment (this is done inside the Penguin Agent Script). Simply put, it is data needed for the penguin to train properly. If the penguin does not have access to information such as the distance to the baby, there is no way it can bring the fish back to the baby, for example.

A reasonable approach for determining what information should be included is to consider what would be needed to calculate an analytical solution to the problem, or what would be expected of a human to be able to use to solve the problem.

The *Vector Action Branches Size* indicates how many actions are possible. In this case, the penguin can only move forward and turn around, so there are two. Branch 0 corresponds to that part of code inside the Penguin Agent Script which manages the movement forward of the penguin, and there are two options: do not move and move forward. Likewise, Branch 1 corresponds to the turn movement, and there are three options: turn left, do not turn, and turn right.

Max Steps (Penguin Agent Script) means the agent resets automatically after 5,000 steps. They indicate the passing of generations as the penguin keeps training until it meets the quota specified in the training file (see section 3.1.4).
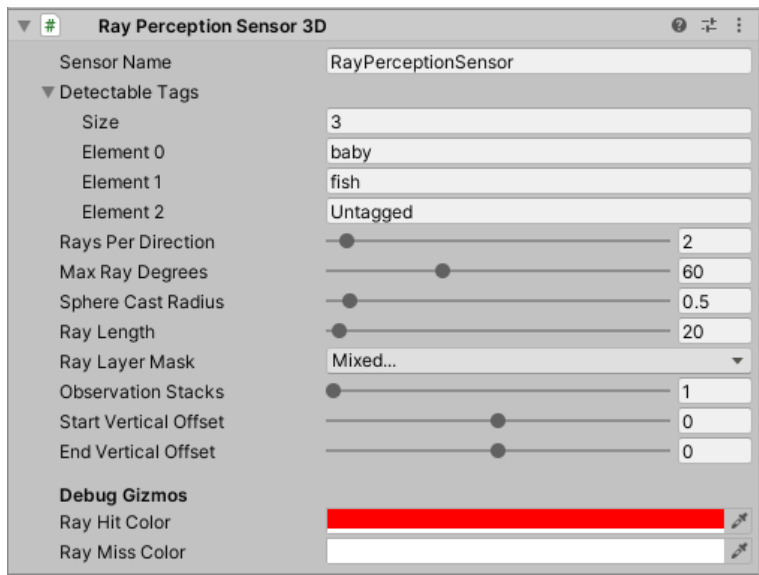
Figure 10: Ray sensor settings.

In addition to having numerical observations, the penguin makes use of another component which is used to make additional observations on the environment; that is *Ray Perception Sensors 3D* (Figure 10).

Detectable Tags tells the sensor which tags to report collisions for. *Rays Per Direction* tells the sensor to cast two rays to either side of centre, as well as straight ahead. *Max Ray Degrees* tells the sensor to spread the two rays out 60 degrees in either direction. The total spread is 120 degrees, 30 degrees between each ray. *Sphere Cast Radius* projects a sphere of radius 0.5 along each ray to test collisions, not just a single point in space.

The screenshot below (Figure 11) shows sphere-casts, two of which have hit the fishes and two of which have hit the walls of the environment. The white line seems to have somehow passed through the wall. This is a reminder that Unity Physics is not flawless, but fortunately ML-Agents is robust enough to work despite occasional misleading or incomplete information.

Figure 11: Screenshot taken directly from Unity's website – Penguin sphere-casts as seen in the Scene view while the game is playing.

Finally, the game was tested to check that everything worked as intended. The test involved controlling the movement of the penguin with the W, A, and D keys on the keyboard, picking up a fish and delivering it to the baby, and checking that the Cumulative Reward text increased as the penguin fed the baby once it was within the six-meter feed radius. When the penguin delivered all the fishes, the area and reward reset.

3.1.4. Training

The next step was to set up the configuration files mentioned earlier. This process was slightly different from the content in the tutorials as newer version of ML-Agents implement different functionalities and/or replace old ones with new ones that are no longer compatible with the older versions. Both configuration files are of type YAML (*Ain't Mark-up Language*) – these files are commonly used for configuration files and in applications where data is being stored or transmitted. One file is always necessary and it contains all the configurations used in training whereas the other file is only used to implement a curriculum to assist in training (Figure 12). A curriculum is normally used to gradually increase the difficulty for the agents and speed up training significantly. In this case, the penguin is rewarded for catching a fish and feeding the baby. Then the curriculum settings will gradually shrink the acceptable feeding radius as the penguin gets better at the task. Without the curriculum, the penguins would take a much longer to learn.
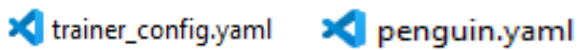
Figure 12: On the left, (trainer_config.yaml) the file containing all the configurations. On the right side, (penguin.yaml) the curriculum.

As mentioned before, the curriculum file needs to match the Behaviour Name (Figure 13) added to the Agent. For example, if the Behaviour Name is "*PenguinLearning*" and the curriculum file is "*PenguinLearn*", training will fail. The way this works is by adding configuration info that the ML-Agents training program uses to control the learning process. Some of these are hyperparameters — values used to control the learning process — and others are settings specific to ML-Agents.
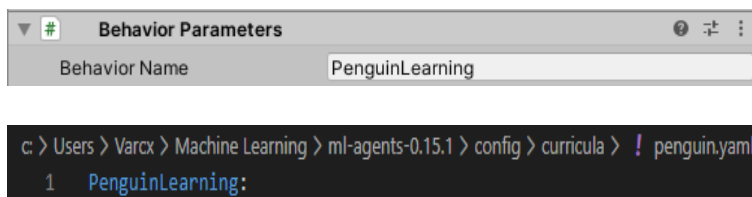

Figure 13: The name in the Behaviour Parameters (Unity Editor) must match the name in the penguin.yaml file.

After configuring the files, the actual training was done using a Python program called mlagents-learn, which was previously installed as part of the Anaconda setup. This program is part of the ML-Agents project and makes training much easier than writing a custom training code from scratch. Anaconda was therefore used to navigate to the ml-agents directory downloaded from GitHub, activate the ml-agents environment and execute the following command to start training:

```
mlagents-learn config/trainer_config.yaml --curriculum
config/curricula/penguin.yaml --run-id penguin_01 –train
```

Here is a breakdown of the different parts of the command:

- *mlagents-learn*: The Python program that runs training
- *config/trainer_config.yaml*: A relative path to the configuration file (this can also be a direct path)
- *--curriculum config/curricula/penguin.yaml*: A relative path to the curriculum files (this can also be a direct path)
- *--run-id penguin_01*: A unique name chosen to give this round of training (it can be anything)
- *--train*: Instructs the program to train the agents rather than just testing them.

The reason that a file is directly passed in for the curriculum, rather than a folder, is that this game only has one Behaviour Name. However, a project may have multiple curriculum files for different Behaviour Names.

Upon running the above command, the following prompt window (Figure 14) appears:


```
Anaconda Prompt (anaconda3)                                                    —    □    X

(ml-agents) C:\Users\Varcx\Machine Learning\ml-agents-0.15.1>mlagents-learn config/trainer_config.yaml --curriculum config/curricula/penguin.yaml --run-id
penguin_02 --train
WARNING:tensorflow:From C:\Users\Varcx\anaconda3\envs\ml-agents\lib\site-packages\tensorflow_core\python\compat\v2_compat.py:65: disable_resource_variables
 (from tensorflow.python.ops.variable_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: 0.15.1,
  ml-agents-envs: 0.15.1,
  Communicator API: 0.15.0,
  TensorFlow: 2.0.1
WARNING:tensorflow:From C:\Users\Varcx\anaconda3\envs\ml-agents\lib\site-packages\tensorflow_core\python\compat\v2_compat.py:65: disable_resource_variables
 (from tensorflow.python.ops.variable_scope) is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
```

Figure 14: Result of running the mlagents-learn program which prompts the user to hit the play button in the Unity editor to get the training started.

As the training proceeds, Anaconda gives periodic updates (Figure 15). Each of these will include:

- Step: The number of timesteps that have elapsed
- Time Elapsed: How much time the training has been running (in real-world time)
- Mean Reward: The average reward (since the last update)
- Std of Reward: The standard deviation of the reward (since the last update).

Figure 15: Update status every 5k steps.

When the mean reward passes a threshold specified in the curriculum file, a new lesson with increased difficulty begins. Eventually, the training reached the maximum curriculum difficulty and the mean reward no longer increased when the penguins could not pick up the fishes any faster. At this point, the training was stopped early by pressing the Play button in the Unity Editor. The training exported a *PenguinLearning.nn* file that represents a trained neural network for the agent (Figure 16).

Note: *This neural network only works for the current penguins. If the observations were to change in the PenguinAgent script or RayPerceptionSensorComponent3D, the neural network would not work.*

Figure 16: The model is saved when the training ends.

3.1.5 Inference

The final next step was to setup the penguins to perform inference. In other words, the penguins make decisions using the neural network previously trained. Once the neural network is setup in the project, Python is no longer needed for intelligent decision making.

As a result, the penguins started catching fish and bringing them to their babies. This is called inference, and it means that the neural network makes the decisions. The agent gives it observations and reacts to the decisions by taking action.

Neural network files are included with the game when you build it. They work across all of Unity's build platforms. If you want to use a neural network in a mobile or console game, you can — no extra installation will be required for the end-users.

3.2. Platformer

As previously mentioned above (see section 3.1), Platformer is based on Penguin. What was learnt when Penguin was built was then applied to Platformer; an original game. Furthermore, unlike Penguin (which is a 3D game) Platformer is a 2D game. Therefore, being a 2D game, the way the system was built and the machine learning implemented is different than Penguin's. In fact, one could argue that, although Platformer is indeed based on Penguin, given the fact that the latter was initially based on an older version of ML-Agents, Platformer evolved into a unique project of its own.

Note: *Since sections 3.1 to 3.1.5 covered most of how machine learning through ML-Agents was implemented in a project, this section and its related sub-sections will treat the subject in a less detailed manner. However, considering how the way machine learning was implemented in both games slightly differs from one another, an explanation highlighting the differences will still be included.*

The very first step in designing Platformer was to find a way to implement machine learning in a 2D game. In order to get an idea on how to structure the game, a research had been carried out to see whether there could be examples of 2D games implementing machine learning. Surprisingly, not valuable content was found. Even after stumbling across the tutorials on Unity's website (see section 3.1), and having learnt how to implement machine learning in a project, there was still a need to find a reference to base the system on.

The need to find an example was not so much about how to structure a 2D game as it was about problem solving; split the problem into smaller problems. The main problem was to build a machine learning using ML-Agents in a 2D game. However, just to create a 2D character controller there is a need to implement left and right movements, and since the game was going to be a platformer, eventually a jump function to implement vertical movement. As a result, the machine learning is also

really different from how it is implemented in Penguin, for example). Therefore, there was a need to simplify the game first.

In that regard, a blog (A Game Developer Learns Machine Learning) was source of inspiration which led to the idea of simplifying the game down to a one-dimensional plane and reducing the gameplay to horizontal movements only.

Note: *the blog has a link to an external GitHub repository containing a 2D platformer game. The temptation of using it as a template for Platformer was really strong (mainly due to the lack of time before the deadline). However, the implications were one too many:*

- *The game uses an older version of ML-Agents,*
- *The project itself is rather big.*
- *Code refactoring would have been a tedious process to go through to make it into something that could be used*

In the end, the GitHub repository was never used, not even as reference material. The inspiration alone on how to simplify the problem was enough to get the project started.

3.2.1. Assets

Unlike the assets in Penguin, which were downloaded externally from Unity's website, Platformer features 2D assets created in GIMP. The default size of each asset is 24x24.
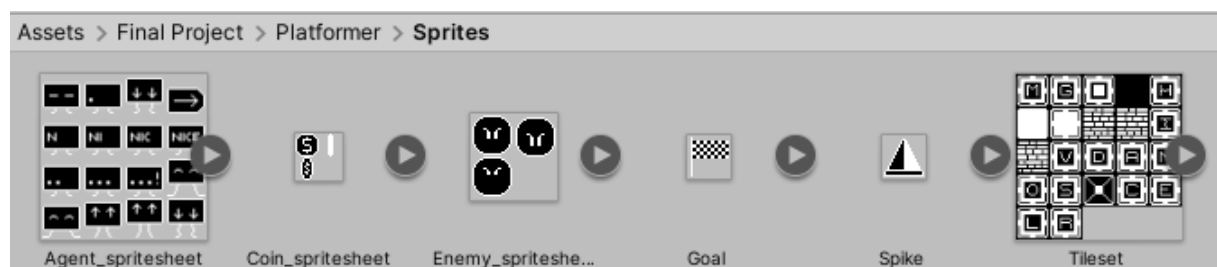
Figure 17: Assets imported in Unity. From left to right: Agent (the main character), Coin (collectable, Agent's secondary objective), Enemy (moving obstacle), Goal (Agent's primary objective), Spike (static obstacle), Tileset (used to create the world map). Except for Goal, Spike and Tileset, all the assets were made into sprite sheet used to create 2D animations.

The assets were first exported as png files and then converted into sprite-sheets by an online converter. The converter takes multiple files and convert them into a single file. The sprite sheets were then imported in Unity (Figure 17) and modified in the Sprite Editor to slice the entire sprite sheet into single sprites again. It might seem a redundant process to go through, but the reason for this is that in terms of size on disk a sprite sheet is more compact than have multiple separate files. Moreover, the sprite sheet is preserved even after being sliced into single sprites.

The game adopts a minimalist pixel-art style. The reason for this choice is due to the fact that such art style is fairly easy to implement and does not require to spend time on the design. As a matter of fact, each sprite was made as a result of random doodling until something pleasing to the eye was created.

3.2.2. One-Dimensional

As mentioned above (refer to section 3.2) the game was initially built as a one-dimensional platformer to create a simplified version of a machine learning first, and then gradually build on top of it to make it two-dimensional.

The agent's objective is to collect coins whilst avoiding the enemies which represent the world's boundaries. The world is a small, enclosed space (Figure 18) so that collecting coins is made easy; the agent does not waste time exploring the environment.
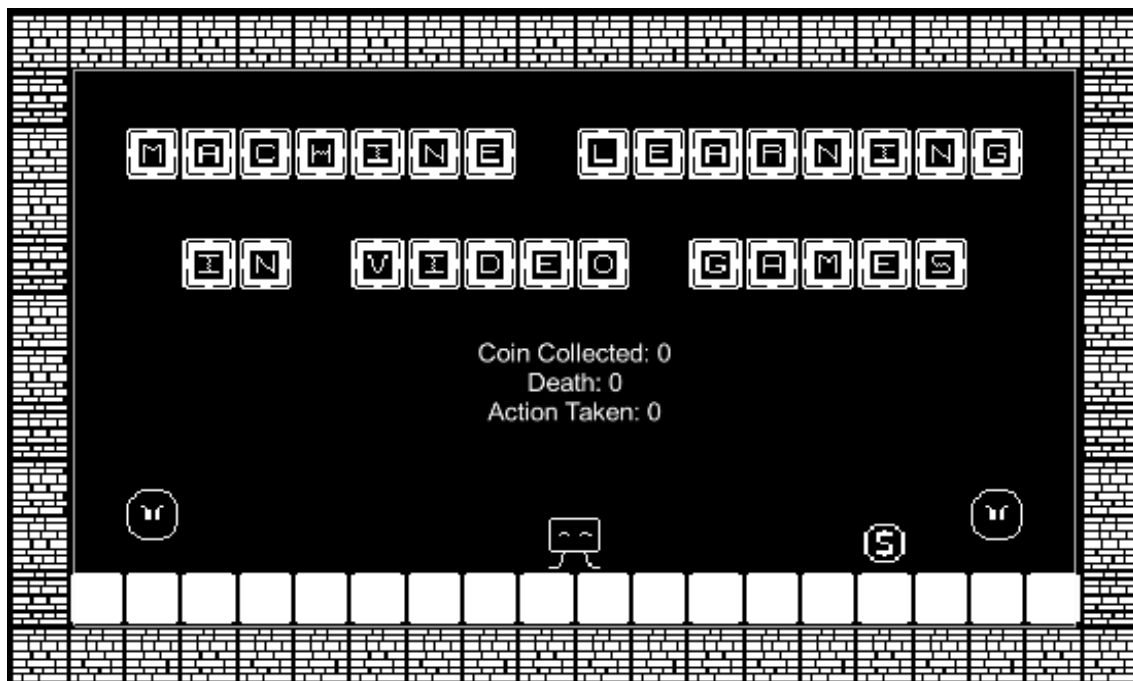
Figure 18: Screenshot of the scene (Platformer1D) seen in Unity. In the middle of the screen a basic HUD displaying basic info for debugging purposes.

The walls act as an insurance in case the agent gets past the enemies which are static obstacles placed on both side of the map.

Note: *Training runs at 100x speed to speed up the training process. Due to the high speed, the obstacles' position was frozen on their X and Z axes to prevent from being pushed back by the agent's jerky movements.*

The agent's position resets upon colliding with the enemies. Likewise, it resets when the agent collects a coin. The coin spawns randomly on the scene within a certain distance between two points that are empty GameObjects, invisibly placed on both left and right side of the scene just before both enemies. The coin's position resets when the agent collides with it and there is no-limit to the number of coins the agent can collect, except for when the agent automatically resets after 5,000 steps during training. Just like in Penguin, the agent is given a small negative reward to encourage action and reach the goal (maximize the mean reward) as quickly as possible. On the other hand, this version of the agent (named SquareAgent1D to make a distinction with its 2D counterpart) does not have a sensor component attached to it as there is need to make additional observations. The Vector Observation Size takes 4 parameters (Figure 19) which are simply the positions of all the element in the scene along the X axis: the agent, the coin and two enemies for a total of 4 elements.
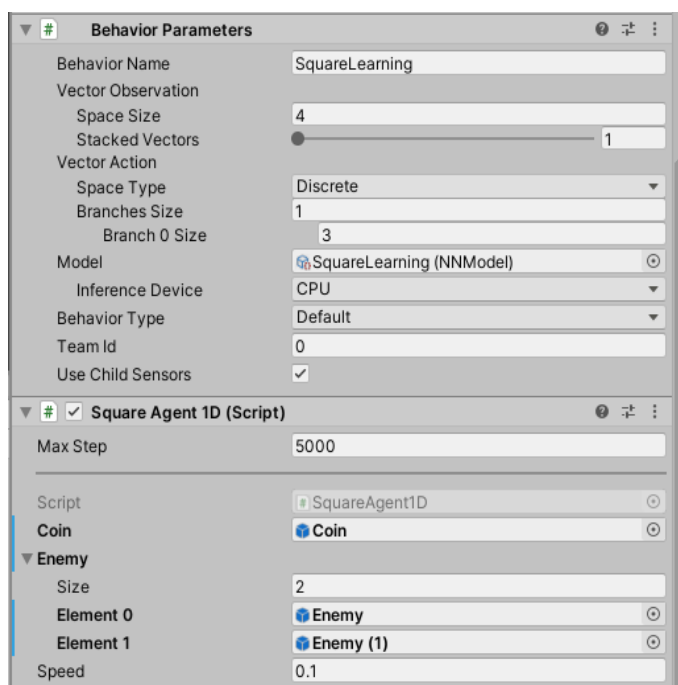
Figure 19: Behaviour Parameters and Square Agent 1D script attached to the Agent object.

Normally, feeding the neural network such observations as the static position of an object is considered bad practice since the agent constantly tries to alter the value in output of these observations by performing different actions. If the value never changes, this may throw off the agent. For instance, observations such as the distance between two points is deemed acceptable since if the agent moves either left or right or even jumps, this value will inevitably change, and if the objective is to get closer to a point, the agent will gradually learn to move one specific direction rather than the other to achieve the desired goal.

Nevertheless, since the scene was built simply as a test to see how to implement a simple machine learning in a one-dimensional plane, the observations were left unchanged.

The Vector Action Size only takes one parameter (Figure 19) which can have 3 possible actions: move left, move right or idle if not moving at all.

3.2.3 Animations

Unlike Penguin, Platformer implements some simple animations to make the game more appealing. In Unity animations are managed by an *Animator Controller* component attached to the GameObject concerned. In Platformer1D (the scene described in the previous section) the agent has only two animations (Figure 20).
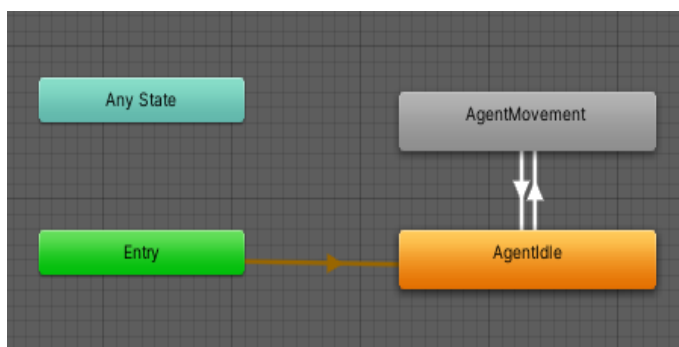
Figure 20: Animation controller attached to the agent in Platformer1D.

Unity's animation system is based on the concept of Animation Clips, which contain information about how objects change their position, rotation, or other properties over time (Figure 21). Animation Clips are then organised into a structured flowchart-like system. The Animator Controller acts as a State Machine which keeps track of which clip should currently be playing, and when the animations should change or blend together.
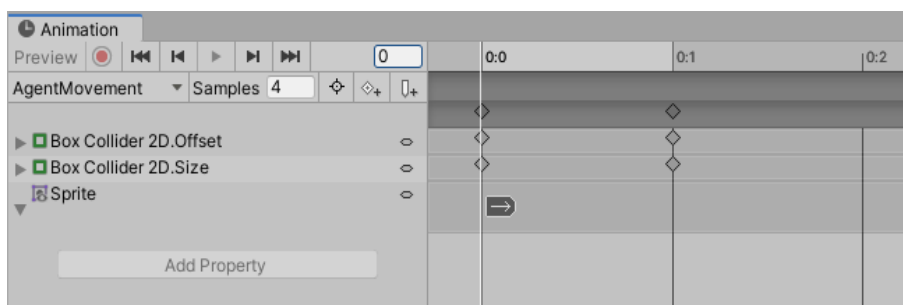
Figure 21: Animation clip of the agent's movement.

The *Entry* node represents which state the state machine begins in. The screenshot above (Figure 20) shows that the agent's entry state is set to *AgentIdle*; an idle animation of the agent when not moving.

This animation will keep playing over time until the state of the object changes and the condition/s to switch to a different animation are met. For instance, if the agent moves left or right, the agent's state will change from AgentIdle to *AgentMovement* and as soon as it stops moving, it will go back to idle.

3.2.4. Training - SquareAgent1D

After implementing SquareAgent1D script to handle the agent's movements and observations, and a HUD to keep track of basic info such as the total number of coins collected, the number of total deaths and the action being taken at the time, the training was handled in the same way it was handled in Penguin. The only difference was that Platformer1D did not need to have multiple game areas to speed up the process considering the simplicity of the problem the machine learning was trying to solve.
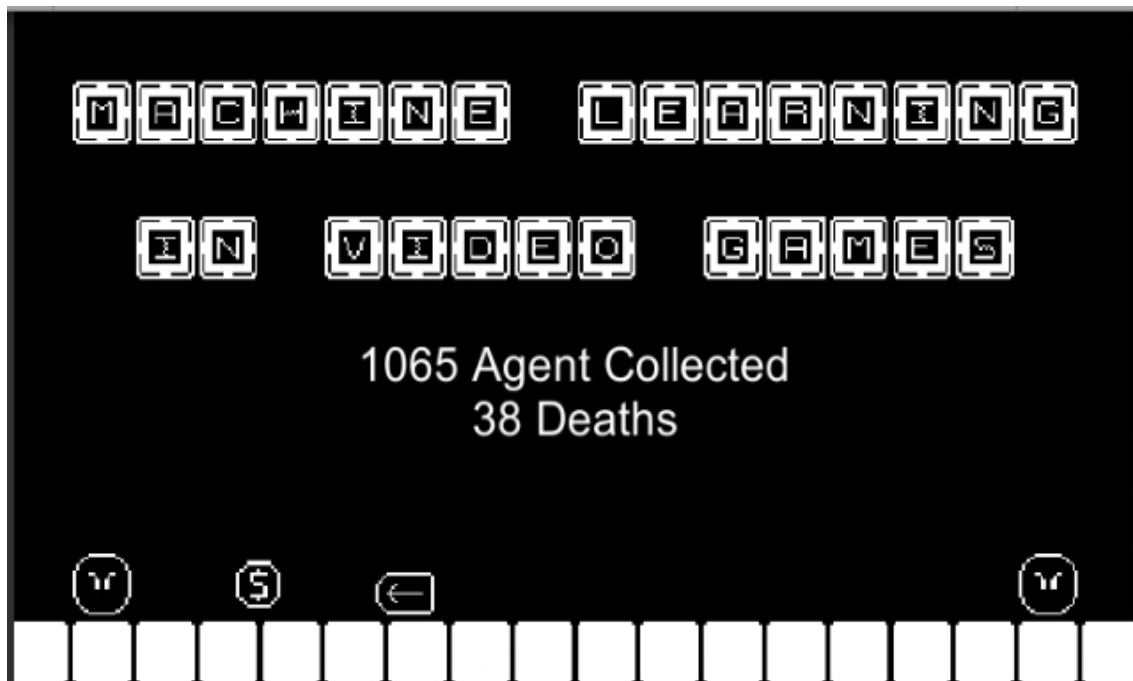
Figure 22: The agent solved the problem. Notice how the HUD does not implement the actions being executed at the time. This is an old version of the scene which did not have all the functionalities.

As a matter of fact, after finding the optimal hyperparameter settings, the agent was able to solve the problem (Figure 22) after only 38 deaths.

3.2.5. Two-Dimensional

To upgrade the game to include 2D functionalities what was done for Platformer1D was then expanded and applied to a new scene; Platformer2D. The scripts and all the components used in Platformer1D were expanded to include extra functionalities. A distinction was made between the assets (scripts and prefabs) used in Platformer1D and the ones used in Platformer2D to keep the

project clean and to keep track of the changes that had been implemented without going to modifying any existing element (Figure 23).
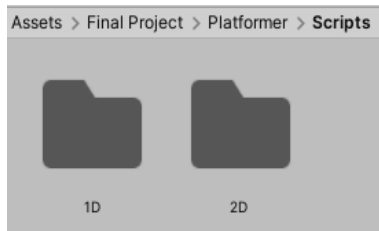
Figure 23: The assets used in Platformer1D and Platformer2D are separated into two different folders.

A jump function was then added (SquareAgent2D script) as a new mechanic to implemented a two-dimensional space. This was done by attaching a child object *GroundCheck* to the agent. As the name implies, this object is needed to check whether the agent is either airborne or touching the ground. If the agent is touching the ground and it is not already airborne, then the agent can jump and a jump animation will play accordingly as long as the agent is in the air. Each element in the scene is assigned a layer (set to *Default* if not changed) which represents group of objects that share a particular characteristic. The GroundCheck takes a parameter that checks whether the layer the agent is colliding with is walkable or not; by default, walkable surfaces are set to *Default*. For example, the enemy has an *Enemy* layer and as such, it is not considered a walkable surface (Figure 24).
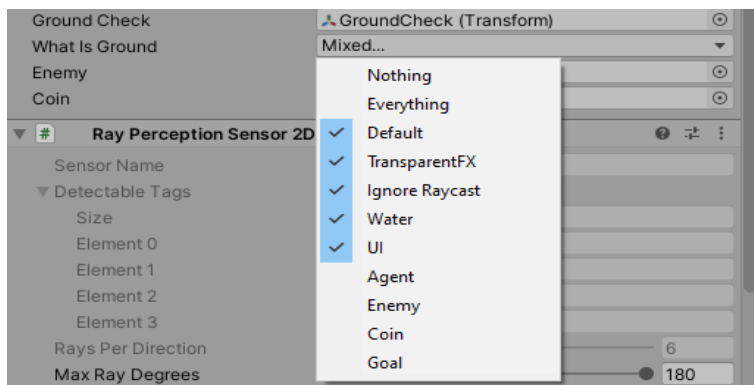
Figure 24: What Is Ground tells the agent the layers it can walk on.

This time, considering the higher complexity of the problem (even adding as a simple mechanic as jumping, makes it for a more complicated) additional observations (aside the ones provided inside the SquareAgent2D script) are provided to the agent through the *RaySensorPerception2D* component. As a matter of fact, the objective of the game is no longer limited to collecting coins, which is now secondary; the game features a new objective: the agent needs to reach an object representing the goal.

The new implemented scene adds a number of new features:

- A list of collectable coins

- Moving enemy

- Spikes

- New objective – the flag

- Improved HUD which includes Jump as an action.

The HUD reflects the way a method inside the SquareAgent2D script works in which an action is associated to a number. This number is then displayed in the HUD along the corresponding action, for example:

Move: 0 Idle, 1 Right, 2 Left.

The reward system was then updated according to the new elements added in the scene. By default, the agent's position resets after 10 seconds after which a negative reward is applied. If the agent collects a coin, however, the timer resets and the agent will have another 10 seconds to collect the next coin. This is to encourage the agent to collect coins as well as reach the goal. Unlike. This kind of replaces the tiny negative reward applied in Platformer1D to encourage to finish the task as quickly as possible. Additionally, the distance between the agent and the goal is calculated and divided into sections in which the reward slightly increases as the agent gets closer to the goal (Figure 25). The agent fails in the following scenarios:

- The agent collides with the enemy

- The agent collides with a spike

- The agent falls off the map

In the last-mentioned case, a collider enclosing the scene represents the world bounds which prevents the agent from falling indefinitely. The agent's position will reset and the death count increase.
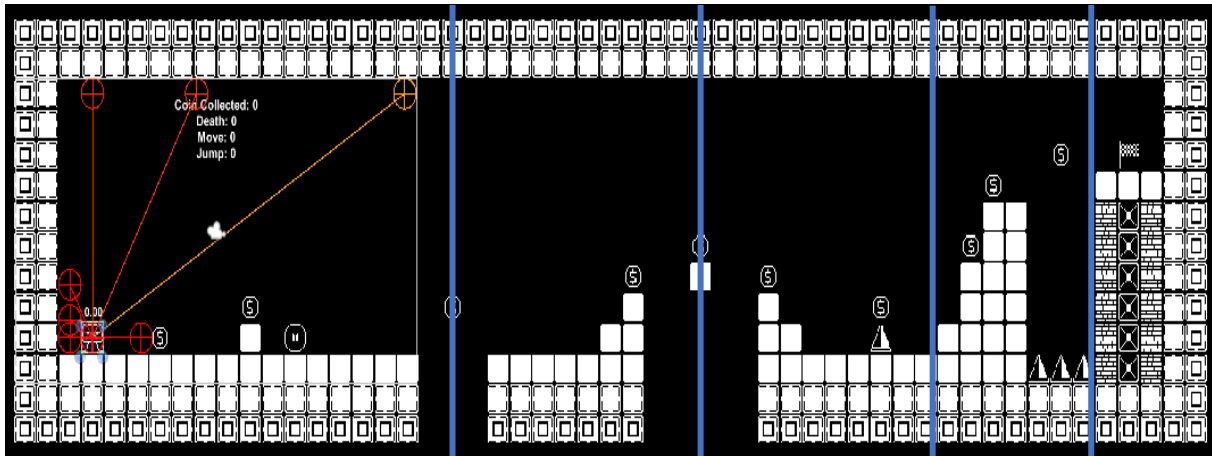
Figure 25: The new scene created to implement the jump function. The blue lines represent the different sections in which the reward increases as the agents gets closer to the goal (on the far right).

The above scene was then made into a training ground where a total of eight game areas were instantiated to speed up training. Furthermore, each agent was given a different colour to distinguish it one from the other. Each game area can be individually selected and viewed whilst the game is running thanks to a script which switches between cameras with the up and down arrow keys.

3.2.6. Training - SquareAgent2D

Once again, the training was done following the same procedure followed in both Penguin and Platformer1D; Anaconda was used to start the mlagents-learn program, the configurations files were modified through a trial and error process to find the optimal parameters. This time, however, a new tool was used for data collection and analysis purposes: TensorBoard (see section 2.3). In order to activate TensorBoard, a new instance of Anaconda needs to be launched. This new prompt needs to run alongside the mlagents-learn program to keep track of the changes the agent goes through and report them in the form of a graph which is viewable in a browser by connecting to the server TensorBoard is hosted on (Figure 26).


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

(ml-agents) C:\Users\Varcx>cd C:\Users\Varcx\Machine Learning\ml-agents-0.15.1

(ml-agents) C:\Users\Varcx\Machine Learning\ml-agents-0.15.1>tensorboard --logdir=summaries
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.0.2 at http://localhost:6006/ (Press CTRL+C to quit)
```

Figure 26: The URL to view TensorBoard related to the system.

TensorBoard is mainly divided into two sections: *Scalars* and *Text*. The former provides graphs on different categories of machine learning such as *Environment*, *Losses* and *Policy*. These categories are further divided into sub categories. For the Environment section there are three graphs: *Cumulative*

Reward, *Episode Length* and *Lesson.* The First graph keeps track of the cumulative reward of the agent which is also displayed in Anaconda (as previously shown in Figure 15). The second graph, Episode length, keeps track of the length of the training session (known as run) which is defined in the configuration files (Figure 27).
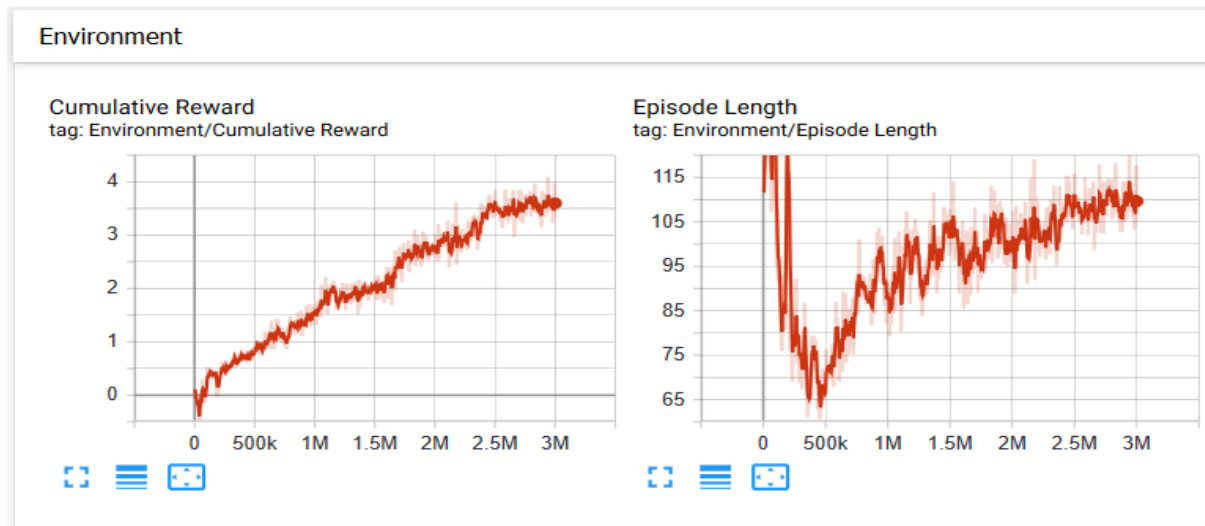
Figure 27: Summaries of an almost-successful run.

Once this reaches the number indicated in the configuration file, the training automatically exits and the game stops running. The last graph is only displayed if a machine learning makes use of a curriculum. This graph updates every time the agent learns a new lesson. For the Losses section there are two graphs: *Policy Loss* and *Value Loss.* Policy Loss indicates how the process for deciding actions changes. The magnitude of this should decrease during a successful training. Value Loss correlates to how well the model is able to predict the value of each state. It increases if the agent is learning and then decreases once the reward stabilizes. Last but not least, the Policy section. In this section there are a total of four graphs: *Entropy, Extrinsic Reward, Extrinsic Value Estimate* and *Learning Rate*. Extrinsic Reward and Extrinsic Value Estimate are related, in a way, to the cumulative mean reward received from the environment. Entropy indicates how random the decision of the models are (Figure 28).
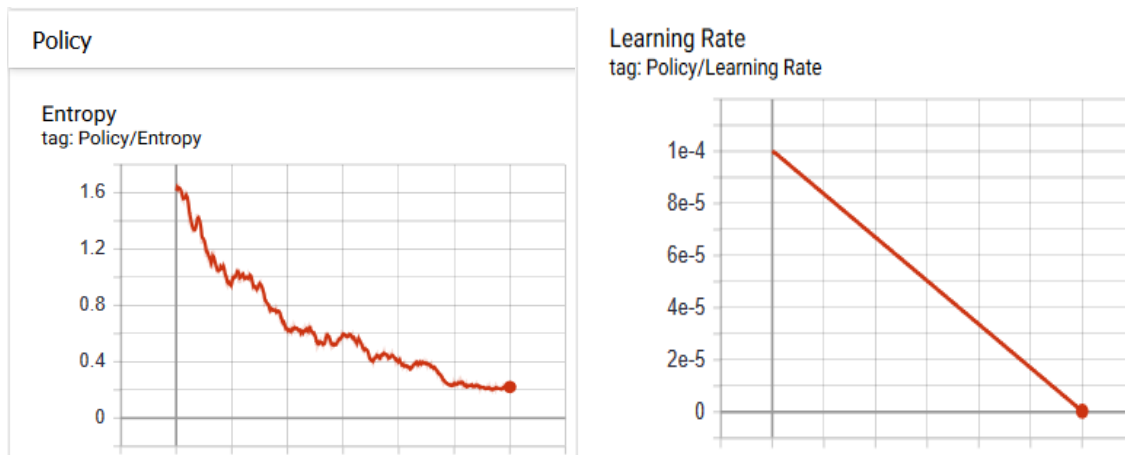
Figure 28: The Entropy and Learning rate of the same run as shown in figure 27. The Entropy showing the expected.

If this value increases or decreases too quickly, the beta (see section 4 for more details) hyperparameter needs to be increased or decreased accordingly. These graphs will show different results depending on the hyperparameters specified in the configuration files. The Text section shows what hyperparameters are being currently used by the model during training.

4. Unity ML-Agents Toolkit

This section aims to briefly outline the kind of technology used in the system described thus far. It provides a brief explanation to cast some light on aspects of this paper that may have to do with technical terminology.

The Unity ML-Agents Toolkit is an open-source project that enables games such as the one discussed in this paper to serve as environments for training intelligent agents. Agents in this system are trained using reinforcement learning. More specifically they use PPO (Proximal Policy Optimization) however, the toolkit provides other machine learning methods as well. The way ML-Agents works is by training the behaviours of Agents by defining three entities at every moment of the game.

- Observations – These can be numeric and/or visual. The Agents in the system discussed in this paper only make use of numeric observation
- Actions – These represent what the agent can take. They can either be continuous or discrete. Discrete actions are normally recommended in a simple environment. If the environment is more complex continuous actions are more appropriate.
- Reward Signals – A value indicating how well the agent is performing. This is normally given only when the agent performs an action that is good or bad.

5. Results

During the training phase the agent would often encounter anomalies as a result of the actions taken. For example, in the screenshot below, despite the agent has fallen off the map, it is still able to bunny-hop through the level whilst trying to reach the goal. This bug is likely caused by GroundCheck which is a small invisible sphere placed in the middle of the agent's feet. This bug makes it so that when the agent lands on the ground as a result of falling down whilst airborne, despite the GroundCheck returns true as soon as the agent hits the ground, it still takes a few milliseconds to settle on the ground. Imagine a person landing on the sand and their feet sink in slightly immediately after as the result of their feet settling on the sand. The weird part is that it may seem like the agent is able to bunny-hop on the box collider representing the world bounds (Figure 29). However, this is not possible considering that the box collider is given a layer *Enemy* to differentiate it from what the agent knows as walkable surface (see Figure 24, section 3.2.5 for more details). Furthermore, if the agent dies as soon as it collides with the world bounds.
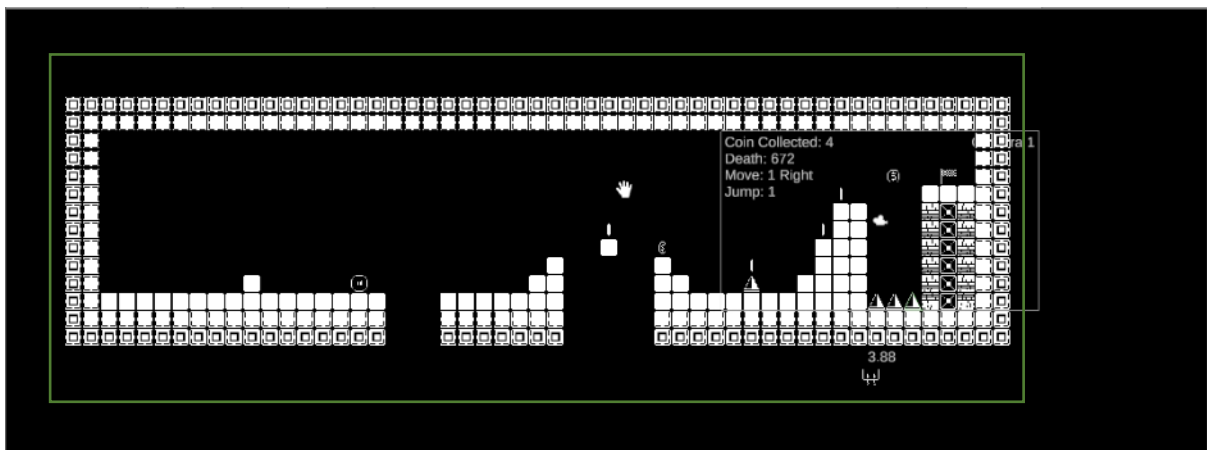
Figure 29: The green outline showing the box collider enclosing the game area.

Nevertheless, the results prove that the agent is indeed able to find glitches in the system.

6. Critical Review

This section reflects the author's personal opinions on the project. It is a critical review involving both his successes and failures. Therefore, the language used in this section is from a personal point of view which aims to advice the reader on how the author felt throughout the entire duration of the project. Last but not least, this review contains areas for improvement, emphasise what has been learnt and how this would affect future projects.

6.1. Project Idea

When I first had to come up with an idea for the final year project, all I could think of was to develop a game. I never really had the chance to create a proper game with all the elements a game normally has: mechanics, graphics, animations, sounds etc. I only ever created small games which had some of the aforementioned elements but not all of them. However, I did not have a defined objective in mind such as solve a certain problem or improve an existing system with additional functionalities or review an old system and come up with a new improved solution. Despite the fact that all the modules I had studied until then had to do, to some degree, with video game development, I had no purpose and no valid explanation to back up my idea of wanting to make a game. When I came up with *Machine Learning In Video Games*, the idea simply came out of the blue. After spending a lot of time thinking of a plausible, viable and yet creative idea for a final year project, this thought crossed my mind: "What if machine learning could be used as a debugging tool in a videogame?". Not only would I have covered a wide range of topics included in the modules I had studied until then as well as module I was going to study during the year, but I would have found a purpose and something to prove. Was it possible to do? I was fully aware that I was never going to build a fully functional machine learning able to debug a game and print the data in some sort of debug log, but if I was somehow able to prove that a working machine learning could spot glitches in a game such as weird hitbox behaviours that trigger some unexpected results (which was the case for this project), then I could call it a success.

6.2. Background Research

So, I started doing some research on machine learning as well as if there ever had ever been past examples of machine learning used for the same purpose. To cover the theoretical part of the project, I borrowed two books from the library as shown in the two screenshots below:

Figure 30: The books I borrowed from the library.

I am ashamed to admit that although I kept them with me for several months (Figure 30), I only read but a few chapters of each book. Not to mention, despite the fact that I gained some insight on the

subject, I never really had to apply the knowledge I had learnt from these books thanks to ML-Agents Toolkit. Furthermore, the reason why I was never able to read them was mainly due to the fact that at the time I was dealing which private issues which hindered my ability to focus on the project. In fact, I put the project on hold for the first half of the academic year and only after I had met my supervisor for the first time in late February, was I able to get started on the project.

Aside from the books, I carried out various online researches to find out if the idea I was going for had already been implemented before. I was able find a very interesting blog but sadly the link to the website (https://www.prowler.io/blog/ai-tools-for-automated-game-testing) was made unavailable sometime later. Nevertheless, this blog demonstrated how a well-trained agent could be used to test design choices and see whether these choices would have a negative impact on the performance of the game. The article explained how something as simple as adding an extra wall, in a 3D space, would imply more lights and shadows, thus increasing the CPU usage. In the end I only read the blog out of curiosity, just to see if my idea had been implemented before, and the answer was yes.

Another important step in my research was when I had to find an alternative to the series of tutorials I used to implement ML-Agent whose contents were obsolete. Luckily, I stumbled across a website (Reinforcement Learning - Penguins) which contained the same series of tutorials but with an updated version of ML-Agents. I was therefore able to use that as a reference to migrate from the version explained in the website (0.14) to the version I had downloaded (0.15). From there on, I was able to build my system and any further research only involved finding solutions to code-related bugs in Unity.

6.4 Planning and Designing

The planning aspect of the project as mentioned at the beginning of this report (see section 2 and 2.1) was greatly neglected. I could have fabricated some sort of fake planning just to show that I had done something, but aside from what I did when I submitted my interim planning and investigation report, I never had the time to implement anything else. Not to mention, I wanted it to keep it real: project management should be implemented in the early stages and further planning be a result of meetings and discussion, which unfortunately was not the case. For these reasons, I did not make any later adjustment to the project management since it would not be a faithful representation of how the project actually went.

	A	B	C	D	E	F
1						
2		User Story / Task / Requirement / Deliverable	Priority	Sprint planned	How long (hours)	Finished?
3		Project Proposal	M	1	4	Y
4		Interim Planning and Investigation Report	M	2	16	Y
5		Arrange VIVA	M	2	/	N
6		Research machine learning	H	3	36	N
7		Learn Python	TBD	3		N
8		Research neural networks	H			N
9		Research supervised learning	H			N
10		Research unsupervised learning	H			N
11		Design neural network	M			N
12		Neural network training	M			N
13		Create AI	M			N
14		Create game level	M			N
15		Write report	M			N
16						

G	H	I	J
Issues with it?			Priority
Took longer than expected to come up with a plausible idea for a project		L	Low
Had a few difficulty to pinpoint what approach to go for		M	Medium
/		H	High
Only just began		M	Must
Might have to learn a different programming language or use one I know of		TBD	To Be Decided
Not yet started			
Not yet started			
Not yet started			
Not yet started			
Not yet started			
Not yet started			
Not yet started			
Not yet started			

Figure 31: Original planning made during the Interim planning and Investigation phase.

User Story / Task / Requirement / Deliverable	Priority	Sprint planned	How long (hours)	Finished?
Project Proposal	M	1	4	Y
Interim Planning and Investigation Report	M	2	16	Y
Arrange VIVA	M	2	/	y
Research machine learning	H	3	36	Y
Learn Python	TBD	3	0	N
Research neural networks	H	/	0	N
Research supervised learning	H	/	20	Y
Research unsupervised learning	H	/	0	N
Design neural network	M	/	0	N
Neural network training	M	/	114	Y
Create AI	M	/	86	Y
Create game level	M	/	65	Y
Write report	M	/	91	Y
TOTAL HOURS			432	

Issues with it?
Took longer than expected to come up with a plausible idea for a project
Had a few difficulty to pinpoint what approach to go for
Was supposed to be in December; Took place in late February
/
I did not really need to learn Python in the end thanks to ML-Agents Toolkit
/
Not really; I research PPO (Proximal Policy Optimization through ML-Agents Toolkit
/
I did not really need to design it in the end thanks to ML-Agents Toolkit
A lot of trial and error to find the optimal hyparameters
Just a few setbacks here and there which I was eventually able to solve.
Issues with level designs
Took longer than expected; I had to ask for an extension

Priority	
L	Low
M	Medium
H	High
M	Must
TBD	To Be Decided

Figure 32: Final version of the planning.

Some of the tasks were abandoned as the project progressed since there was no longer need to do them (Figure 31 and 32). For example, I never really had to research unsupervised learning when I found out I could implement my system using ML-Agents Toolkit. The total amount of hours is based on the following calculus: from 15th April to 26th May for approx. 8 hours a day = 328 hrs to design, code and complete the project; from 26th May to 8th June (after I was granted an extension due to COVID-19) for approx. 8 hours a day = 104 hrs. This gives a total of approximately 432 hours. This is just an estimate; some days I studied for 12 hours straight so the numbers above are just indicative of the total amount of hours spent on the project.

The project went a bit astray from what I had initially envisioned. I wanted to make a test case scenario in which two instances of the same game would run at the same time in a horizontal split-screen like manner so that one could see the differences between the two different implementations. The one at the top being a level well designed in which the agent is able to complete the level effortlessly, and the one at the bottom being a level intentionally designed to fail (e.g. the agent cannot jump the gap between two platforms that are too distant one from another). Nevertheless, the main core of the project has been left unchanged throughout the development process and I am quite proud of it.

6.3. Improvements

The system features an additional level called Level1 which is double the size of the previous levels. Although it features the same elements, the agent struggles to complete it. This is probably due to the reward system which should be improved in order to make the model more generic. For one, the distance between the agent and the goal is hard coded, meaning that the sections in which the reward gets higher as the agent gets closer to the goal is calculated with numbers. Simply put, this should be replaced by percentages: 100 to 0; 100 being the farthest from the goal and 0 the closest. If the agent

is 70% away from the goal then the reward increases +0.1, if the agent is 40% then +0.3 and so forth. If this was implemented then no matter how big the map would be, the sections representing the various distances that trigger different rewards would be evenly distributed. For two, I am quite positive the observations need to be revisited. The agent still struggles to understand that the goal is the ultimate objective which gives the best reward.

Another improvement would be further hyperparameter-tuning to improve the efficiency of the model in Platformer2D which would definitely help solve Level1. In other words, to make the model as generic as possible so that it can solve many different problems within the same context. At some point in the project I had implemented a curriculum for the agent in Platformer2D. This definitely improved the learning rate of the agent; however, it made the model less generic due to the type of curriculum it was given. The agent was given a countdown of 10 seconds and made aware of it in its observations. As a result, the agent would associate certain actions to the countdown. For example, if the agent had to jump to avoid an obstacle and the number displayed on the countdown was five, it would associate the number five to jumping. So, the agent understood that to avoid that particular obstacle it would need to jump when the countdown reached five. However, when the agent tried to solve Level1, it had previously learnt to always jump when the countdown reached five, but since the map was different, and the obstacle which could be avoided when the countdown reached 5 was no longer there, replaced by a pit, the agent would jump straight into the pit, no second thoughts given. Therefore, that type of curriculum made the model only able to solve a specific problem.

Minor improvements would be to implement sounds effect to make the game a bit more engaging, even though it is not a playable game. Therefore, I thought this was a useless feature to have in a game that is mainly used for testing purposes.

6.4. Lessons Learned

Undertaking this project has made me realize that I am lacking in the management skills department. Despite the fact that I have had issues which affected my ability to progress with my project the way I had initially envisioned, setbacks happen all the time and I should have been able to deal with them in a better way.

I completely disregarded planning because I was months behind with the schedule and plunged myself into the development aspect of the system, unaware of the possible consequences this approach would have on my project. The engagement with the project process was kept to a minimum as a consequence of my falling behind and I failed to keep my supervisor up to date on my project status. Luckily, I was still able to create a working system of my own relying only on my abilities, but I am fully aware that project management was one of the main aspects of this project.

In future projects I want to be able to create a thorough project plan, considering all the variables (possible setbacks) first and then build on top of it week by week or month by month by setting new tasks and giving myself deadlines to improve myself in making accurate time estimates.

7. References

Analytics Vidhya. 2020. Commonly Used Machine Learning Algorithms | Data Science. [ONLINE] Available at: https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/. [Accessed 26 March 2020].

Andrew Tch. 2020. The mostly complete chart of Neural Networks, explained. [ONLINE] Available at: https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464. [Accessed 27 March 2020].

Toptal Engineering Blog. 2020. PyTorch Reinforcement Learning: Teaching AI How to Play Flappy Bird | Toptal. [ONLINE] Available at: https://www.toptal.com/deep-learning/pytorch-reinforcement-learning-tutorial. [Accessed 29 March 2020].

GameDev.net. 2020. The Total Beginner's Guide to Game AI - Artificial Intelligence - Tutorials - GameDev.net. [ONLINE] Available at: https://www.gamedev.net/articles/programming/artificial-intelligence/the-total-beginners-guide-to-game-ai-r4942/. [Accessed 30 March 2020].

Practical Artificial Intelligence. 2020. Teaching an AI to play a simple game using Q-learning - Practical Artificial Intelligence. [ONLINE] Available at: https://www.practicalai.io/teaching-ai-play-simple-game-using-q-learning/. [Accessed 30 March 2020].

Mauro Comi. 2020. How to teach AI to play Games: Deep Reinforcement Learning. [ONLINE] Available at: https://towardsdatascience.com/how-to-teach-an-ai-to-play-games-deep-reinforcement-learning-28f9b920440a. [Accessed 5 April 2020].

GitHub. 2020. GitHub - Unity-Technologies/ml-agents: Unity Machine Learning Agents Toolkit. [ONLINE] Available at: https://github.com/Unity-Technologies/ml-agents. [Accessed 15 April 2020].

GitHub. 2020. ml-agents/Using-Tensorboard.md at master · Unity-Technologies/ml-agents · GitHub. [ONLINE] Available at: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Using-Tensorboard.md. [Accessed 16 April 2020].

Unity Learn. 2020. ML-Agents: Penguins - Unity Learn. [ONLINE] Available at: https://learn.unity.com/project/ml-agents-penguins. [Accessed 17 April 2020].

GitHub. 2020. ml-agents/Training-ML-Agents.md at master · Unity-Technologies/ml-agents · GitHub. [ONLINE] Available at: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-ML-Agents.md. [Accessed 17 April 2020].

GitHub. 2020. Release ML-Agents Beta 0.15.1 · Unity-Technologies/ml-agents · GitHub. [ONLINE] Available at: https://github.com/Unity-Technologies/ml-agents/releases/tag/0.15.1. [Accessed 17 April 2020].

GitHub. 2020. ml-agents/Installation.md at master · Unity-Technologies/ml-agents · GitHub. [ONLINE] Available at: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Installation.md. [Accessed 18 April 2020].

A Game Developer Learns Machine Learning - Intent | MikeCann.co.uk. 2020. A Game Developer Learns Machine Learning - Intent | MikeCann.co.uk. [ONLINE] Available at: https://mikecann.co.uk/machine-learning/a-game-developer-learns-machine-learning-intent. [Accessed 18 April 2020].

Unity3D.College. 2020. Unity3D Machine Learning - Writing a custom Agent - Create your own AI Bot - Unity3D.College. [ONLINE] Available at: https://unity3d.college/2017/11/01/unity3d-machine-learning-writing-a-custom-agent-create-your-own-ai-bot/. [Accessed 18 April 2020].

GitHub. 2020. ml-agents/Getting-Started.md at master · Unity-Technologies/ml-agents · GitHub. [ONLINE] Available at: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Getting-Started.md. [Accessed 18 April 2020].

GitHub. 2020. ml-agents/Learning-Environment-Design.md at master · Unity-Technologies/ml-agents · GitHub. [ONLINE] Available at: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Learning-Environment-Design.md. [Accessed 18 April 2020].

GitHub. 2020. ml-agents/Learning-Environment-Design-Agents.md at master · Unity-Technologies/ml-agents · GitHub. [ONLINE] Available at: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Learning-Environment-Design-Agents.md. [Accessed 19 April 2020].

Simple Timer - Unity Answers. 2020. Simple Timer - Unity Answers. [ONLINE] Available at: https://answers.unity.com/questions/351420/simple-timer-1.html. [Accessed 20 April 2020].

how to round a float to 2 DP - Unity Answers. 2020. how to round a float to 2 DP - Unity Answers. [ONLINE] Available at: https://answers.unity.com/questions/50391/how-to-round-a-float-to-2-dp.html. [Accessed 20 April 2020].

The timer starts after a collision with an object - Unity Answers. 2020. The timer starts after a collision with an object - Unity Answers. [ONLINE] Available at: https://answers.unity.com/questions/1293287/the-timer-starts-after-a-collision-with-an-object.html. [Accessed 20 April 2020].

Stack Overflow. 2020. C# - How to Stop Timer when User Collides with an Object? (Unity) - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/51159063/how-to-stop-timer-when-user-collides-with-an-object-unity. [Accessed 20 April 2020].

Immersive Limit. 2020. Reinforcement Learning Penguins (Part 1/4) | Unity ML-Agents — Immersive Limit. [ONLINE] Available at: https://www.immersivelimit.com/tutorials/reinforcement-learning-penguins-part-1-unity-ml-agents. [Accessed 25 April 2020].

GitHub. 2020. ml-agents/Training-Configuration-File.md at master · Unity-Technologies/ml-agents · GitHub. [ONLINE] Available at: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-Configuration-File.md. [Accessed 25 April 2020].

YouTube. 2020. 2D World building w/ Tilemap & Cinemachine for 2D - Random and Animated Tiles [6/8] Live 2017/22/08 - YouTube. [ONLINE] Available at: https://www.youtube.com/watch?v=6JXAAsVIVbs&list=PLX2vGYjWbI0RX4rWxhg15UJ9GGf3tFLNQ&index=6. [Accessed 3 May 2020].

YouTube. 2020. 2D Platformer Character Controller - Player Controller Script [6/8] Live 2017/2/22 - YouTube. [ONLINE] Available at: https://www.youtube.com/watch?v=pJah2FmrXkk&list=PLX2vGYjWbI0SUWwVPCERK88Qw8hpjEGd8&index=6. [Accessed 3 May 2020].

Unity Learn. 2020. Movement Basics - Unity Learn. [ONLINE] Available at: https://learn.unity.com/tutorial/movement-basics?projectId=5c514956edbc2a002069467c#. [Accessed 5 May 2020].

YouTube. 2020. Unity 5 2D Platformer Tutorial - Part 5 - Improvements & Fixes. - YouTube. [ONLINE] Available at: https://www.youtube.com/watch?v=l22ovXJf0R0&list=PLq3pyCh4J1B2va_ftIthSpUaQH0LycRA-&index=5. [Accessed 6 May 2020].

Stack Overflow. 2020. c# - Conversion of System.Array to List - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/1603170/conversion-of-system-array-to-list. [Accessed 15 May 2020].

Removing objects from an array - Unity Answers. 2020. Removing objects from an array - Unity Answers. [ONLINE] Available at: https://answers.unity.com/questions/339083/removing-objects-from-an-array.html. [Accessed 17 May 2020].

Remove an object from an array and destroy it (C#) - Unity Answers. 2020. Remove an object from an array and destroy it (C#) - Unity Answers. [ONLINE] Available at: https://answers.unity.com/questions/204819/remove-an-object-from-an-array-and-destroy-it-c.html. [Accessed 17 May 2020].

Destroy OTHER objects on Collision Unity - Unity Answers. 2020. Destroy OTHER objects on Collision Unity - Unity Answers. [ONLINE] Available at: https://answers.unity.com/questions/1077612/destroy-other-objects-on-collision-unity.html. [Accessed 17 May 2020].

YouTube. 2020. Hyperparameter Tuning for Unity ML-Agents. [ONLINE] Available at: https://www.youtube.com/watch?v=ZKzXAVp8bC8. [Accessed 22 May 2020].

Unity Technologies. 2020. Unity - Manual: Animation System Overview. [ONLINE] Available at: https://docs.unity3d.com/Manual/AnimationOverview.html. [Accessed 26 May 2020].

Unity Technologies. 2020. Unity - Manual: State Machine Transitions. [ONLINE] Available at: https://docs.unity3d.com/Manual/StateMachineTransitions.html. [Accessed 26 May 2020].

OpenAI. 2020. Proximal Policy Optimization. [ONLINE] Available at: https://openai.com/blog/openai-baselines-ppo/. [Accessed 08 June 2020].

https://www.prowler.io/blog/ai-tools-for-automated-game-testing (no longer available)

Total word count (excluding references): 9,602.

Appendix 1

The following screenshots show the engagement with the project which was kept to a minimum due to the circumstances explained in previous sections of the paper (see section 2, 2.1, 6 and its relative sub-sections).

Daniele Di Modica (student)
Sat 14/12/2019 17:16
To: Stelios Kapetanakis; Karl Cox

Hi,

I am Daniele, a third year student of Computer Science Games, and I am writing this email regarding my individual project (Machine Learning In Video-games) which has been appointed to your supervision.

First of all, I am sorry I have not been able to introduce myself personally yet.

After I took time off university last year (due to private matters), I have been extremely busy trying to catch up with the coursework and as a result, I completely forgot about the project Viva. I am fully aware it fell under my responsibility and I am truly sorry for it.

As far as I understand, the Viva was supposed to be completed by 13th Dec. Is there any chance that I can still arrange it for us to discuss the project? If so, what date and time work best for you?

I look forward to hearing from you.

Yours faithfully,

Daniele.

Daniele Di Modica (student)
Fri 21/02/2020 14:10
To: Stelios Kapetanakis <S.Kapetanakis@brighton.ac.uk>

Hi Stelios,
I'm currently on my way to uni. There's been a problem with the bus I'm on I reckon I'll be there in 10 more mins.
I'd still like to meet if possible

. . .

Stelios Kapetanakis
Hi Deniele, I had a message from Karl and I am afraid he cannot make it today. We can still meet if you want or we can aim for next week that he will be more flexible. Let me know Best Stelios   Fri 21/02/2020 12:55

⌄ See 4 more messages

Stelios Kapetanakis
Hi Deniele, How are you! How is your project going? Best Stelios   Tue 11/02/2020 10:53

Daniele Di Modica (student)
Fri 01/05/2020 18:04
To: Stelios Kapetanakis

Machine Learning.rar
unibrightonac-my.sharepoint.com

Hello Stelios,

I am finally replying to the message you sent me a while ago, I believe it was on Teams, concerning my final project.

First of all, although I feel like I am just repeating myself at this point, I want to apologize for disappearing on you (for the second time).
When I saw your message, especially after our first and last meeting, I did not have the courage to tell you that I hadn't made much progress with the project. I had literally nothing to show.

But now I can finally reply to you and say that the project has entered its final stage. I do not know if what I have done will be considered enough. For now, I just want to brief you on what I have been doing since the last time we spoke.

In the end, due to my own lack of organisation and poor time management, I decided to stick to my first idea about making a 2D platformer video-game where an AI agent plays the game.

The idea was to build a small level and have the Agent "debug" it on my behalf.
In other words, I wanted to see if it was possible to use machine learning as a debugging tool that could replace the role of a game tester.

And I am excited to say that I think it is viable.

I started off by researching 2D video-games that used machine learning, just to get an idea on how I could possibly structure my project.
I was surprised to find little to no information on the internet.

I was already set to work in Unity, as I was already quite familiar with it and I did not have time to explore different options anyway. That is when I stumbled across a 3D tutorial in Unity that made use of reinforcement learning. So, I decided to make that my starting point.

Machine learning in Unity can be done by using the ml-agents toolkit which is a plugin that enables games to serve as environments for training agents and it uses a simple-to-use Python API.

So to cut the story short, I went through the tutorial, I grew accustomed to this ml-agents toolkit, and then I proceeded by creating my own 2D game.

The tools I have been using are the following:
- Unity
- Gimp
- ML-Agents Toolkit
- Anaconda
- TensorFlow
- Visual Studio Code
- Visual Studio 2017
- Excel

The very first step was to think of my problem in its simplest form.
And for me that resulted in creating a 1D game where the agent only needed to move left and right.

After that, I took it up a notch and made it 2D.

What really baffled me was that even though I had played the game over and over to make sure that everything worked as intended (despite being aware of certain bugs) when I finally let the agent play the game, I was super excited to see it was able to spot those issues I knew of and even find new ones I never thought they could exist.

The final result is an agent that makes use of PPO (Proximal Policy Optimization) to complete a small level that has the following features:

- The agent is able to move left and right and jump
- Moving enemy
- A goal
- Collectibles to encourage the agent take actions all the way to the goal

And this is where I am at now. Lately, I have been busy doing some hyper-parameters tuning to find the optimal settings and optimize learning.

I am positive I have proved the point of my project to some extent: machine learning is a viable tool for level design testing.

I say to some extent because my implementation of machine learning, in its current state and due to my lack of coding skill, does not have a way to actually detect possible glitches within the game.
The only way I have right now, is to record the training session and then watch it and see if there is any anomaly, which still requires human intervention and presence. However, theoretically speaking, someone with better skills than me or even a company would be able to implement a debug log system with all the inputs used by the agent or some visual aid to facilitate debugging. So that you only need to look at those, and then fix the issues within the code.

but I do intend to write more about this (including pros and cons) in my report so that I cover bits about further areas for possible investigations or enhancements.

I look forward to hearing from you soon!

Kind regards,

Daniele

P.S. I have attached a few screenshots and a small video (OneDrive link) to help you better understand the nature of my project and what it looks like.

Stelios Kapetanakis Chat Files Organisation 1 more ∨ +

Stelios Kapetanakis 19/03 09:26
Hi Daniele, let me know how your project is going. Do you have any question for me

12 May 2020

12/05 12:14
Hi Stelios, it's been a while

The project is going fairly well, I daresay

As I mentioned in my email from a couple of weeks ago, it has entered the final stage where I have been doing some testing to make sure that everything works as intended

Stelios Kapetanakis 12/05 13:59
that is great, thanks Daniele.

12/05 14:07
Sorry for not getting back to you sooner

I did mention that in my email as well 🙂

17 May 2020

17/05 16:39
Hi Stelios, do you know if it is possible to ask for an extension even for the final project? I might need that extra week to write the report, after all.

18 May 2020

Stelios Kapetanakis 18/05 13:42
Hi Daniele, mitigating circumstances may be the best way to go forwards

Appendix 2

Link to OneDrive to view Assets/Code of the project: https://unibrightonac-my.sharepoint.com/:f:/g/personal/d_dimodica1_uni_brighton_ac_uk/EgQMEOGqGKtCp1tn6bkwFOwBcm_n93iiQkfzwSQd0Di2qw?e=nTcGi5

Instructions to open the project:

- The project was made in Unity 2019.3.15f1. Therefore, it is recommended to open the project using the same or a newer version of Unity.
- The Assets zip file contains the assets used in the project which are divided into two folders.
  - The Assets folders contains GIMP file which are the files made to create the 2D games
  - The fbx folder contains the files downloaded form Unity's website and used for Penguin
- Training through Anaconda will not be needed as each agent has a trained model (neural network model).
- Simply unzip the Machine Learning zip file and import it into the Unity Editor.
- Navigate to the Scene folder and Select any scene you wish to view
- The agent in Level1 does not have a NN model and therefore it can only be manually controlled.

Appendix 3

Link to Project Presentation: https://youtu.be/IR2DL0svSP4

Appendix 4: Further TensorBoard Graphs

The following screenshows are summaries of various unsuccessful runs (that implemented a curriculum) which were exited early at 500k step and one successful run which lasted until the end. This was decided purely on the fact that at 500k, the successful run (blue line, square2d_15_SquareLearning2D) showed the most progess among all the previous runs.
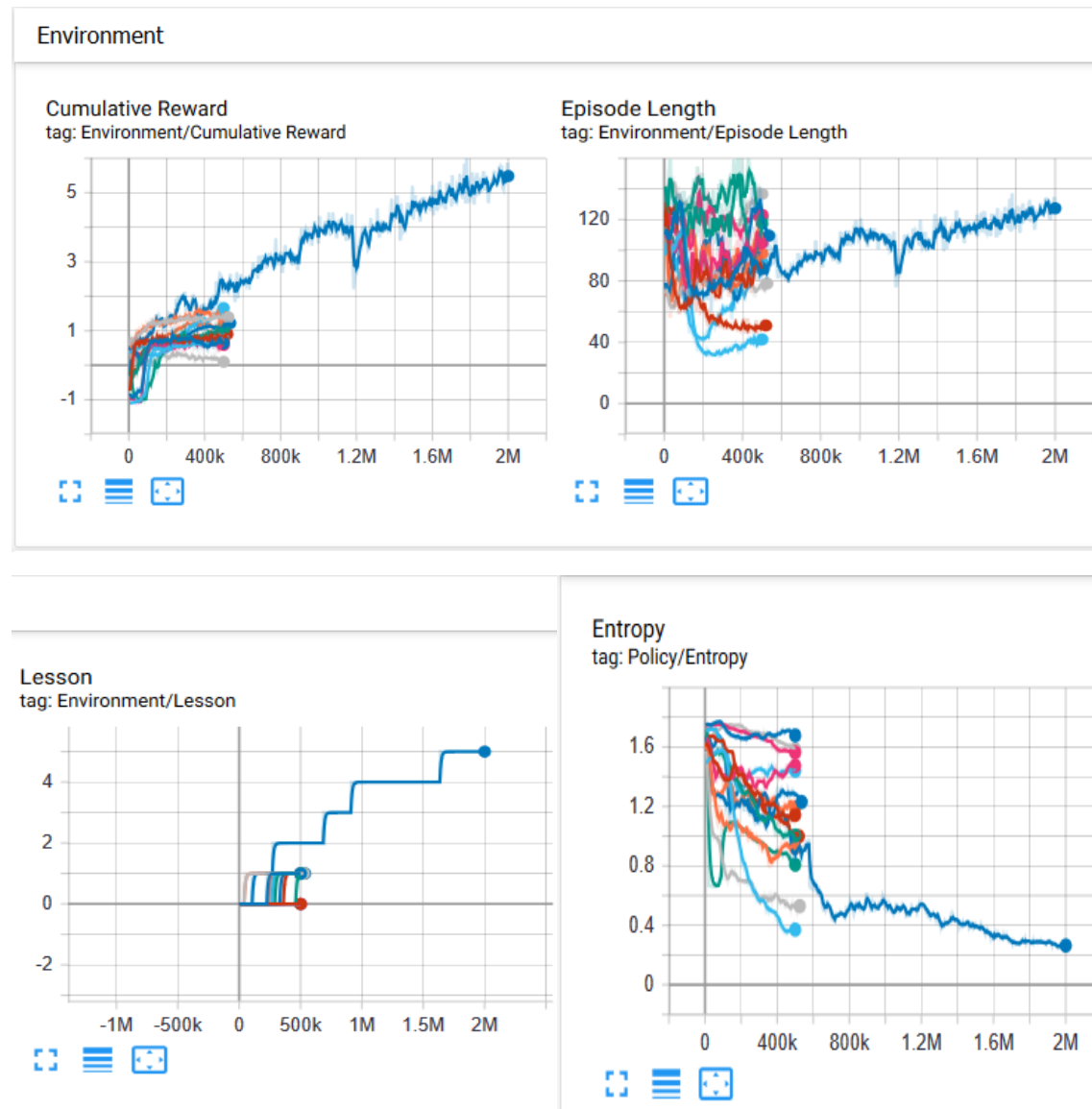
Runs

Write a regex to filter runs

☑ ◯ square2d_01_SquareLearning2D
☑ ◯ square2d_02_SquareLearning2D
☑ ◯ square2d_03_SquareLearning2D
☑ ◯ square2d_04_SquareLearning2D
☑ ◯ square2d_05_SquareLearning2D
☑ ◯ square2d_06_SquareLearning2D
☑ ◯ square2d_07_SquareLearning2D
☑ ◯ square2d_08_SquareLearning2D
☑ ◯ square2d_09_SquareLearning2D
☑ ◯ square2d_10_SquareLearning2D
☑ ◯ square2d_11_SquareLearning2D
☑ ◯ square2d_12_SquareLearning2D

TOGGLE ALL RUNS

summaries

☑ ◯ square2d_13_SquareLearning2D
☑ ◯ square2d_14_SquareLearning2D
☑ ◯ square2d_15_SquareLearning2D

TOGGLE ALL RUNS

summaries