

An Investigation into Machine Learning through the Simulation of Human Survival

Computer Science NEA

Name:

Candidate Number:

Centre Name: Barton Peveril College

Centre Number: 58231

Contents

1	Analysis	4
1.1	Statement of Investigation	4
1.2	Background	4
1.3	Expert	4
1.4	Initial Research	5
1.4.1	First Interview	5
1.4.2	Existing Investigations	6
	Crafter	6
	Minecraft	7
	Conway's Game of Life	8
1.4.3	Algorithms and Potential Data Types	9
	Neural Network and Matrices	9
	Procedural Generation	10
	Proposed Programming Language and Associated Libraries	11
1.5	Prototype	13
1.5.1	Prototype Objectives	13
1.5.2	Terrain Generation and Displaying to Window	13
	UpDownNeutralGen Method	15
	Average Method	16
1.5.3	Finished Terrain Generation	17
1.5.4	Matrix Data Structure	17
1.5.5	Prototype Evaluation	20
1.6	Second Interview	20
1.7	Objectives	22
1.8	Modelling of the Problem	23
1.8.1	Matrices	24
	Overview	24
	Matrix Addition	24
	Matrix Subtraction	24
	Matrix Multiplication	24
	Matrix Scalar Multiplication	25
	Matrix Hadamard Product	25
	Matrix Transpose	25
1.8.2	Forward Propagation	25
	Overview	25
	Pre-Activation	25
	Activation	26
	ReLU	26
	Leaky ReLu	27
	Sigmoid	27
	TanH	27
	SoftMax	28
1.8.3	Differentiation	28
	Differentiation from First Principles	28
	Standard Differentiation Rules	28
	Chain Rule	29
	Partial Derivatives	29
1.8.4	Back Propagation	29
	Overview	29
	The Bellman Equation	29

	Loss Function	30
	Gradient Descent	31
	Differentiating Activation Functions	31
	Simple Network	33
	Complex Network	34
2	Design	36
2.1	System Flow Charts	36
2.2	Class Diagrams	37
2.3	Individual Classes	39
2.4	Choice of Programming Language and Libraries	45
2.5	Description of Algorithms	45
2.5.1	Matrix Addition	45
2.5.2	Matrix Subtraction	46
2.5.3	Matrix Multiplication	46
2.5.4	Matrix Scalar Multiplication	46
2.5.5	Matrix Hadamard Product	47
2.5.6	Matrix Power	47
2.5.7	Matrix Transpose	47
2.5.8	Activation Function SoftMax	48
2.5.9	Neural Network Forward Propagation	48
2.5.10	Half Square Difference	48
2.5.11	Neural Network Bellman Equation	49
2.5.12	Neural Network Backwards Propagation	49
2.5.13	Experience Replay	49
2.5.14	Agent Get Tile Vector	50
2.5.15	Agent Convert to Grayscale	50
2.5.16	Agent Post Process Tile Vector	50
2.5.17	Agent Spawn Position	51
2.5.18	Enemy Spawn Position	51
2.5.19	Enemy Move	52
2.5.20	Poisson Disc Sampling	52
2.5.21	Perlin Noise	53
2.5.22	Octave Perlin Noise	54
2.5.23	Heap Heapify	54
2.5.24	Heap Extraction	55
2.5.25	Heap Sort	55
2.6	Description of Data Structures	55
2.7	File Structure	56
3	Testing	59
3.1	Testing Table	59
3.1.1	Targetted Testing Areas	59
3.1.2	User Input and Program Output Tests	60
3.1.3	Matrix Implementation Tests	60
3.1.4	Deep Reinforcement Learning Algorithm Tests	62
3.1.5	Data Logger Tests	63
3.1.6	Simulation Tests	63
3.2	Testing Evidence	65
3.2.1	User Input and Program Output Evidence	65
3.2.2	Matrix Implementation Tests	70
3.2.3	Deep Reinforcement Learning Algorithm Evidence	76

3.2.4	Data Logger Evidence	78
3.2.5	Simulation Evidence	81
4	Evaluation	84
4.1	Evaluation of Objectives	84
4.1.1	Reading user inputted data	84
4.1.2	Generating the Environment	84
4.1.3	Displaying the world to a Pygame Window	84
4.1.4	Simple Agent with a set of Actions	84
4.1.5	Matrix Class with Standard Operations	84
4.1.6	Creation of a Reinforcement Learning Model	85
4.1.7	Creation of a Data Logger	85
4.2	Answering the Proposed Question	85
4.3	Analysis of Training Data	86
4.4	Expert Feedback	89
4.5	Evaluation of Expert Feedback	90
4.6	System Improvements	90
5	Technical Solution	92
5.1	main.py	92
5.2	simulation.py	92
5.3	newAgent.py	96
5.4	enemy.py	100
5.5	worldClass.py	101
5.6	perlinNoise.py	105
5.7	deepqlearning.py	106
5.8	activations.py	112
5.9	datalogger.py	114
5.10	heap.py	116
5.11	plotData.py	117

1 Analysis

1.1 Statement of Investigation

I plan to investigate Machine Learning and Neural Networks by developing a Survival Simulation environment in which a character will be controlled by a Machine Learning algorithm.

The Machine Learning Algorithm I choose to implement will most likely require lots of Complex Maths, from prior knowledge I know that Matrices and Calculus are heavily used within Neural Networks. Most of this Maths I will have covered in my Maths and Further Maths Lessons, but some will require independent research on my Part.

The survival simulation will be procedurally generated and present multiple challenges towards this character in order to provide a complex problem for it to solve. The procedural generation will be based upon a seed, and will generate Terrain which the character has to explore and navigate. The challenges could be things like collecting items, or having to avoid/kill enemies which are actively tracking the character and trying to hinder its progress.

The key question I aim to answer with this investigation is:

Can I develop a Machine Learning Algorithm to survive in a pseudorandom, open-world environment?

This question is rather broad and allows me to experiment with varying levels of complexity when it comes to the simulation. As long as I can implement a Machine Learning Algorithm which is complex enough to solve the provided challenges.

1.2 Background

I am investigating Machine Learning because I've been wanting to try my hand at it for a while, this Project will allow me to gain a broad understanding of Neural Networks and their applications, along with an understanding of procedural generation. Machine Learning is an evolving field, with mere infinite applications from Image Recognition to Self Driving Cars.

Old

I am investigating this area of Computer Science because I've been interesting in attempting a form of Machine Learning for a while now but haven't had a reason to dive into it. Machine Learning is an evolving field, with mere infinite applications such as Image Recognition, Chat Bots, Self Driving Cars, etc. I feel as though my project will be sufficiently advanced enough to expand my knowledge of the subject. It will require lots of research, planning, and design work in order to successfully fulfil my Technical Solution.

1.3 Expert

For my expert I approached one of my friends, Shaun, who has prior experience with Machine Learning. He has created his own Hand Written Digit Recognition Network before, along with using Python Libraries such as *PyTorch* to train an agent to play the game *Flappy Bird*, among other ML projects. He has a much better understanding of Machine Learning than me currently, so hopefully he will be a good resource as I develop my project.

1.4 Initial Research

1.4.1 First Interview

As part of my Investigation I approached my friend Shaun, who has Experience with Machine Learning. I formed a list of questions to ask him, the responses are paraphrased for clarity. I mainly wanted to gain an idea of what Machine Learning algorithm would suit my project the best. So I targetted my questions towards this.

1. What are your first impressions of my project?

“Your project is definitely very complex and if finished will tick alot of the boxes needed for Full Marks. There are lots of layers of complexity along with room for good Object Orientated Design.”

2. What Machine Learning Algorithms do you think would be relevant to my project?

”Without pushing your complexity too far I think you should look into Deep Reinforcement Learning, I believe it has the possibility of solving your problem if not too complex. Because of that you may want to keep your simulation as minimal as possible in order to give your Agent a chance. If you wanted to go further you could implement a Convolutional Neural Network, but this will add to the Complexity and take more time to program.”

3. Would User Defined Parameters be helpful?

”The ability to dynamically change the parameters through a json file or similar would be very useful. Epecially to users who have little to no experience with it before hand. The ability to change things like the Procedural Generation, Enemy Counts, Network Structure etc would be the perfect addition to your project.”

4. What Procedural Generation method would be best for my Project?

”I only have experience with Perlin Noise but I think that it would be a great fit for your Project. It uses simple vector Maths to calculate Gradient Noise, and is relatively simple to understand and Program. There are other Procedural Generation Methods I’m aware of like Diamond Square or Simplex Noise, but both of those are much more complicated to my understanding.”

5. How complex should I make my Simulation?

”I would stick to a relatively simple simulation at first, and then if your agent is successful at solving it, you can add more to test the limits of your network after. Dynamic threats like Enemies which follow the Agent which it can attack would provide a base complex problem to start off with. Other problems could be collecting items or a simple Food Collection system with a Hunger Meter.”

6. How should I determine if my project is successful?

”You could log a graph of Loss compared to Time, and in theory if your agent is learning it will successfully reduce the average Loss the more training it receives. You could use this graphed data as supporting evidence in your Evaluation.”

7. What should I focus my Initial Research on?

"It would be beneficial to you to research the Maths behind Neural Networks, specifically for Forward Propagation and Back Propagation. The Maths behind it can get very complicated, along with being very hard to debug if a small error is made. They both heavily rely on Matrix Operations, so if you're not familiar with those you should get up to speed."

1.4.2 Existing Investigations

Crafter

In my research on the Internet I discovered a project called *Crafter*.

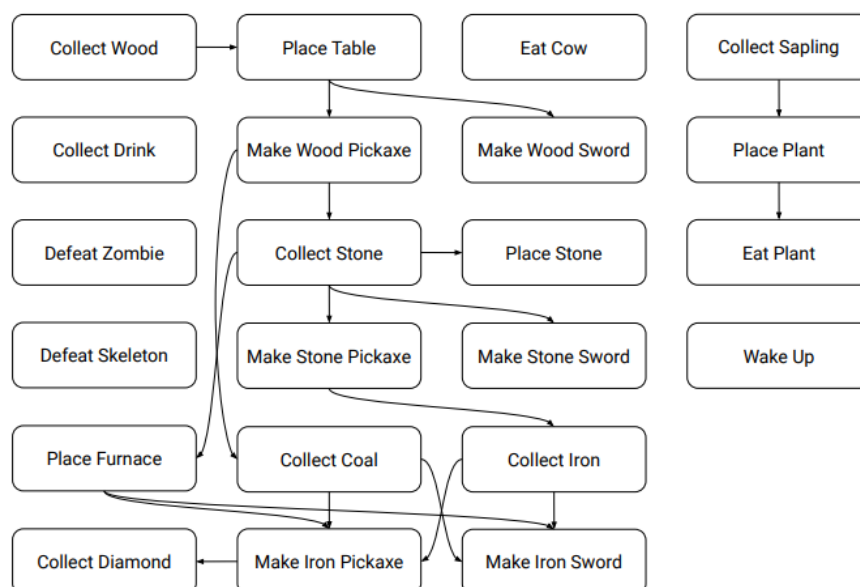
<https://github.com/danijar/crafter>

Crafter is described to be "*Benchmarking the Spectrum of Agent Capabilities*", and is utilised in conjunction with Machine Learning Algorithms such as *DreamerV2*, *PPO* and *Rainbow*. Crafter poses significant challenge towards its Player, requiring high levels of generalisation, long-term reasoning, and complex problem solving. If the Machine Learning algorithm in question fails to achieve one of these aspects it will struggle to full "Solve" the simulation.

High levels of generalisation are required when training a Machine Learning algorithm, if this is not achieved then your network will only lend itself to a single Dataset/Problem. An example of this would be training a network used to recognise hand written digits on only one way of writing 4's, if presented with an input for a different type of 4 it may not recognise it and identify it incorrectly.

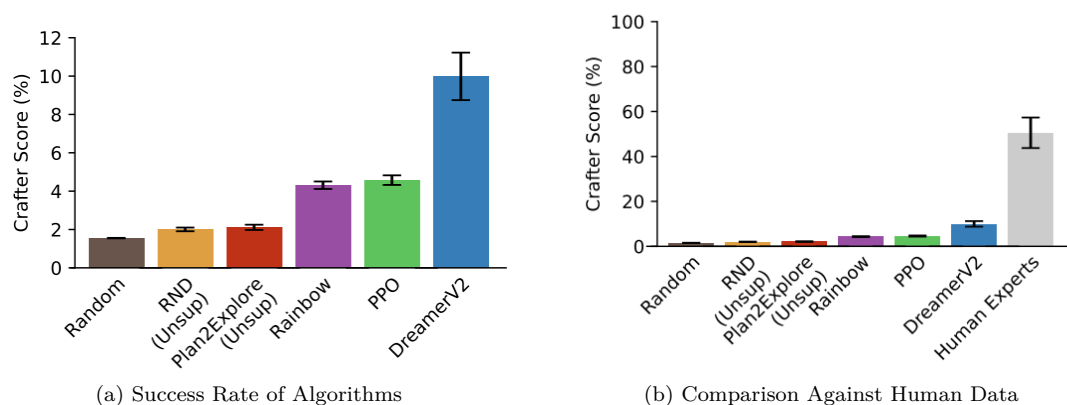
Long-Term reasoning is a complex problem to solve in the context of Machine Learning, current Machine Learning models struggle to deal with this problem. This is dealt with by using algorithms built to mimic "memory". A common implementation of this is Experience Replay which stores states in a queue, and relearns from it after every N ammount of steps.

A complex reward and action system may take time for an algorithm to learn but it certainly is possible with current Machine Learning Models. Crafter utilises a complex action system with a flow chart determining which Action can be taken given the current state of the simulation. Below is shown the Complex Flow Chart of Actions:



Complex action system as shown in the Paper "Benchmarking the Spectrum of Agent Capabilities"

Crafter manages to achieve quite high success rates with various Algorithms, but they still fail to overcome, or even match human standards. This is likely due to the complexity of the problem, and in theory will be solvable within the near future as Machine Learning advances over the next few years. This is why I plan to create a simpler simulation which the Agent will be more likely to be able to solve. Below is shown the Success Rate Data for both Algorithms and Human Experts.



While I would love to create a simulation similar to crafter, it is very complex and would take a long time to develop. Yet would not net many marks in the process. Overall I feel like Crafter is a good example that my project is possible, but will require a complex Machine Learning Model in order to achieve reliable results from my Investigation.

Minecraft

Minecraft is a *very* popular Game. It's a sandbox game, meaning that the player can do almost anything they want. The game is formed from blocks which can be broken or placed, along with a plethora of items, enemies, passive animals and more. It has infinite terrain generation, and explicitly uses Perlin Noise. The seed of the noise determines all the terrain generation, loot tables, random structures, caves, etc.

First it starts off on a very broad level, painting a basic topographical map of the world. It uses Perlin Noise to sample a height value for each chunk, where chunks are 16x16 areas of blocks. Then within these chunks the game uses the Diamond Square algorithm to interpolate between it and the chunks around it, creating blocks where the terrain should be. This produces an entirely deterministic results based upon the seed.

Secondly, the Caves are generated using Perlin Worms, which travel in deterministic directions based on their starting position. These worms dig through the terrain carving out caves which can then be traversed by the player. Within these Caves spawn water sources, pools of lava, useful ores. All of these are deterministically generated by the original seed. Minecraft itself is too complex and dynamic to be solved by current Machine Learning algorithms, along with there is no quantifiable metric for performance due to it's sandbox nature. There exist data sets for Minecraft, in the form of captured gameplay footage, but there has been little to no success of quantifiably good solutions to solving Machine Learning problems within Minecraft.

Overall I feel like it would be good to borrow elements from Minecraft's terrain generation, such as its utilisation of Perlin Noise. But the majority of the games systems are way too complex for a Machine Learning algorithm to solve.



(a) Example of Minecraft's terrain generation in a Swamp Biome



(b) Example of a Sunken Pirate Ship Structure

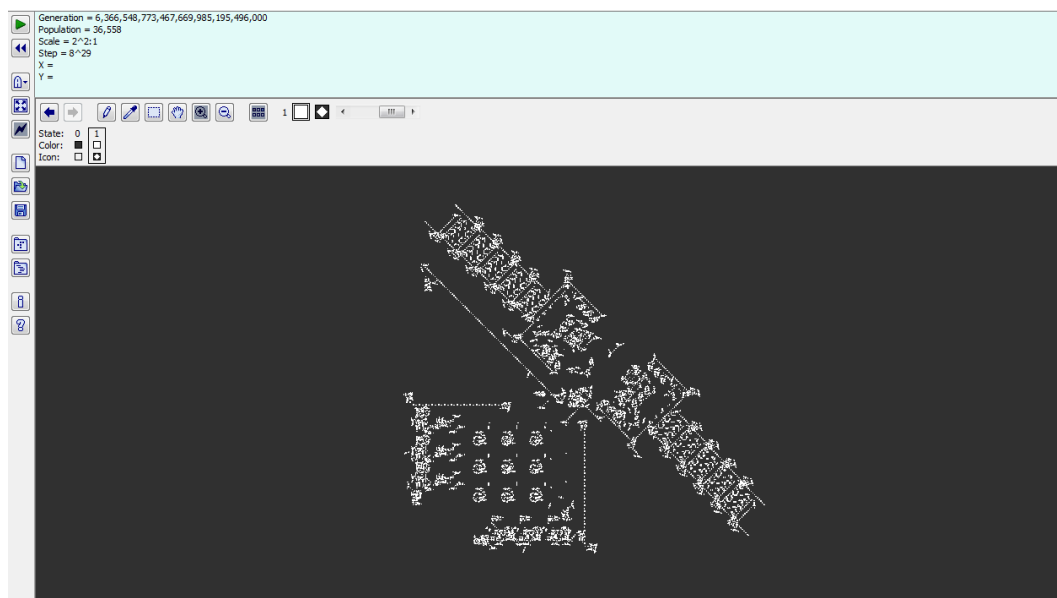
Conway's Game of Life

Conway's Game of Life is what's called a Cellular Automaton, which is a discrete computation model formed from a grid of cells along with a ruleset. Conway's is commonly referred to as a Zero Player Game, where the input for the Automaton is defined at the start, with no further adjustment needed for it to run. The game is fully Turing complete and can simulate a Universal Constructor.

The rules of Conway's are such that:

1. Any live cell with fewer than two live neighbours dies, as if by underpopulation.
2. Any live cell with two or three live neighbours lives on to the next generation.
3. Any live cell with more than three live neighbours dies, as if by overpopulation.
4. Any dead cell with exactly three live neighbours becomes a live cell.

It is rather interesting that such complicated Machines can be formed from such a simple ruleset, as an example here is a Turing Machine formed from 34 Thousand Cells:



(a) Turing Machine built in Conway's Game of Life

Overall, I think this shows that my simulation doesn't need to have complex rules in order to achieve interesting results. Conway's is formed from 4 simple rules, and yet is Turing complete.

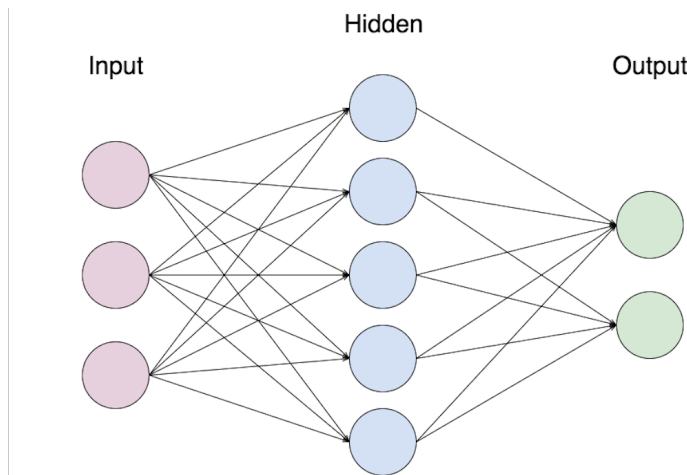
1.4.3 Algorithms and Potential Data Types

Neural Network and Matrices

As part of developing a Machine Learning Algorithm, I will need to implement a Matrix class in order to implement a Neural Network. Matrices are commonly used to represent individual layers of a network. Along with making calculations much easier, condensing them into performing operations on matrices, rather than using nested for loops and lists. As part of my Initial Research I have taken the time to understand how a Neural Network functions, it turns out I have already learned most of the Maths needed to understand how it works in my A Level Maths and Further Maths courses.

A Neural Network functions as a series mathematical equations used to recognise relationships between inputs and desired outputs. They take in a Vector of Input Data, and output a Vector of Output Data. They can be represented in simple terms as a function: $N(x)$ where: $\{x \in V, N(x) \in V\}$, should you adopt the black box approach. The functions name in this case is Forward Propagation.

We form a Neural Network with multiple layers of Nodes, the layers being referred to as the Input Layer, Hidden Layer/s and Output Layer. In this case each Node is connected to every Node in the previous layer and the following layer. In the below image is represented a Neural Network with a layer structure of [3, 5, 2].



Each connection, otherwise known as an Arc or Edge, has an associated weight. Along with every output of a layer having an associated Bias. These are used to compute the outcome of a network.

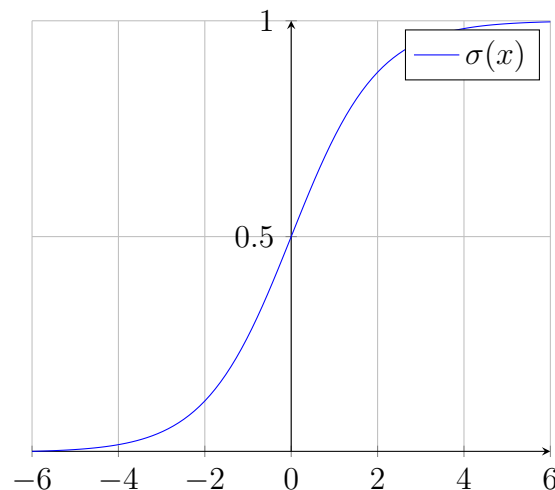
Forward Propagation is used to compute the outcome of a network, it has a general form and uses Matrix Multiplication and Addition to achieve this.

$$S^{(L)} = \begin{bmatrix} s_0^{(L)} \\ s_1^{(L)} \\ \vdots \\ s_n^{(L)} \end{bmatrix} = \begin{bmatrix} w_{0,0}^{(L-1)} & w_{0,1}^{(L-1)} & \cdots & w_{0,m}^{(L-1)} \\ w_{1,0}^{(L-1)} & w_{1,1}^{(L-1)} & \cdots & w_{1,m}^{(L-1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,0}^{(L-1)} & w_{n,1}^{(L-1)} & \cdots & w_{n,m}^{(L-1)} \end{bmatrix} \begin{bmatrix} a_0^{(L-1)} \\ a_1^{(L-1)} \\ \vdots \\ a_n^{(L-1)} \end{bmatrix} + \begin{bmatrix} b_0^{(L)} \\ b_1^{(L)} \\ \vdots \\ b_n^{(L)} \end{bmatrix}$$

$$\sigma(S^{(L)}) = \sigma \left(\begin{bmatrix} s_0^{(L)} \\ s_1^{(L)} \\ \vdots \\ s_n^{(L)} \end{bmatrix} \right) = \begin{bmatrix} \sigma(s_0^{(L)}) \\ \sigma(s_1^{(L)}) \\ \vdots \\ \sigma(s_n^{(L)}) \end{bmatrix}$$

We then apply an activation function as shown above, in this case we will apply the Sigmoid

function: $\sigma(x)$ to $S^{(L)}$. The Sigmoid function is a Mathematical Function which *squishes* values between 0 and 1. Shown Below:



Matrices can be used for all parts of a Neural Network implementation, and will prove very useful in my Technical Solution.

Procedural Generation

For my project I am going to have to procedurally generate 2d terrain, while researching this I came across a few algorithms which seemed to be able to do this pretty well. I will compare two algorithms I discovered below.

Post-Processing Algorithms	Perlin Noise
I discovered two post processing algorithms often used for simple 2d terrain generation. 1 Averages squares around the selected square, and the other pulls it up or down the gradient its currently on. I find these interesting because they're relatively simple, and I'm not quite sure whether they will produce good results or not. So it would be interesting to test out implementing these in my prototype.	Perlin Noise is an algorithm developed by Ken Perlin for use in the digital generation of noise. This noise can be combined to create <i>realistic</i> looking height maps for world generation. Perlin Noise retains continuity and is seeded so the generation can be entirely controlled. By "retains continuity" I mean that you can sample the same point and retrieve the same value. If I was to implement Perlin noise it would take longer, but also might end up with a better result due to it being more widely used. It's a trade-off between time to implement and desired result.

I also discovered an algorithm called Poisson Disc Sampling, this can be used to sample random points in N dimensional space. It takes in 2 values, the R and K value, these values determine the output of the function. The R values is the minimum distance a point has to be from another, randomly placed point which hasn't been selected yet. If the distance between any existing points is less than R, the point will be rejected and another will be selected. The K value determines how many rejected are needed before the algorithm will stop attempting to choose a new point.

Proposed Programming Language and Associated Libraries

When selecting a Programming Language and associated Graphical Libraries I took into consideration a few options. Below I have weighed up 3 options for Programming Language, along with 2 graphical libraries per language

Proposed Solution	Benefits and Downsides of Proposed Solution	
Python	Python is the first thought which comes to mind when I think about programming, it is my favourite language and I'm yet to find anything which I prefer. Its very versatile and great for rapid prototyping, the dynamic typing makes It great for coding quickly without worrying too much about whether you're using a <i>float32</i> or <i>float64</i> . It also has hundreds of libraries and is very well supported by its developers and the community.	
Python Graphical Libraries	Pygame	Pygame is a highly customizable and well developed binding of <i>Simple DirectMedia Layer</i> (SDL) Library. It has a full set of 2d drawing tools, along with keyboard and audio capabilities. I have lots of experience with Pygame so I already have code which I can take from, which will speed up development when dealing with the Pygame library.
	Tkinter	Tkinter provides an interface to the standard <i>Tcl/Tk GUI Toolkit</i> , which is available for most platforms, this makes it highly versatile. Though as my project is not intended as a software package I dont see this as being an incredibly big selling point. Tkinter will serve mostly the same purpose as Pygame but give me easier options for Graphical Input, I dont currently plan to add GUI so this feature isnt neccesary.

Proposed Solution	Benefits and Downsides of Proposed Solution	
C#	C# is my second favourite language, I have plenty of experience with it from developing games with Unity. It's faster than Python and is less abstracted, but this speed isn't necessarily required for my project. With C# I could utilise the <i>Unity Game Engine</i> for my project, but then I might end-up relying on builtin types and functions rather than developing my own.	
C# Graphical Libraries	Windows Forms	Windows Forms is a relatively simple drag drop interface for designing your own applications. I've never used it before but I could utilise it with C# to create my project. I believe it might be a bit overkill for my needs though, as it includes many, many UI features which I will have no use for.
	WPF	WPF or <i>Windows Presentation Foundation</i> is a versatile development platform for desktop applications. It is relatively versatile in its uses and utilises XAML and is the UI Language of Windows Platforms. XAML would be a new language for me to learn but I have experience with HTML so I don't believe it would be too difficult. The platform would provide a stable base to my project.
Proposed Solution	Benefits and Downsides of Proposed Solution	
Rust	Rust is low level language designed for speed and efficiency, I started using it recently as a side hobby and would like to use it more in future projects of mine. Though I feel like it may be a bit overkill for a Computer Science NEA, with it often being used for server side applications rather than general purpose applications.	
Rust Graphical Libraries	Piston2d	Piston2d is a feature complete 2d graphics library which utilises OpenGL, I've worked with it briefly before and I believe it would be a good option over Pixels if I needed more complex drawing methods.
	Pixels	Pixels is a lightweight 2d graphics library designed to simply push pixels to the screen, Its relatively simple and i've used it for making a simple <i>Falling Sand Game</i> before, could be a good little option if I wanted to develop a lightweight solution.

1.5 Prototype

1.5.1 Prototype Objectives

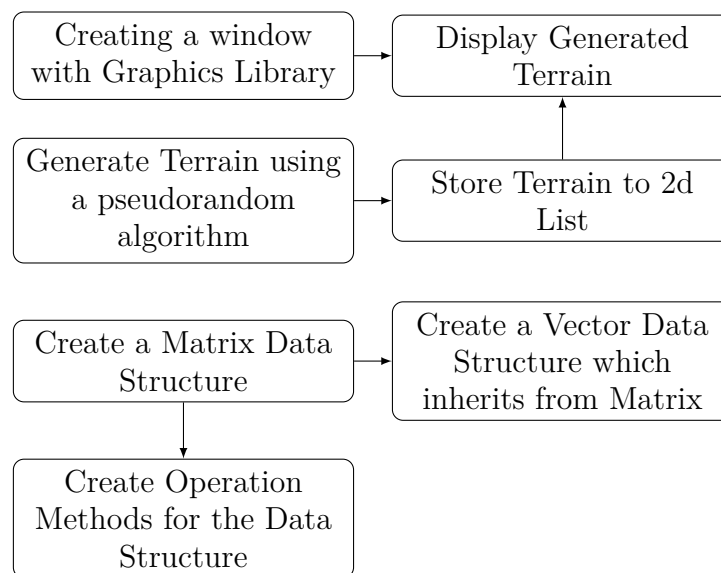
Before starting my Prototype I had to decide upon a short list of objectives I wanted to complete/investigate as part of it. These boiled down to a few things:

- Terrain Generation
- Displaying the Generated Terrain using a Graphics Library
- Matrix and Vector implementation

For my Prototype, I first created a GitHub Repository, available here:

<https://github.com/TheTacBanana/CompSciNEAPrototype>

I had created a hierarchy of importance for development in my head, visualized using this flow diagram:



I decided to use Python for developing my Prototype, this seemed like a good fit due to me having lots of experience with the language. Python is a Dynamically Typed and interpreted which makes it versatile for prototyping and fast, iterative development.

1.5.2 Terrain Generation and Displaying to Window

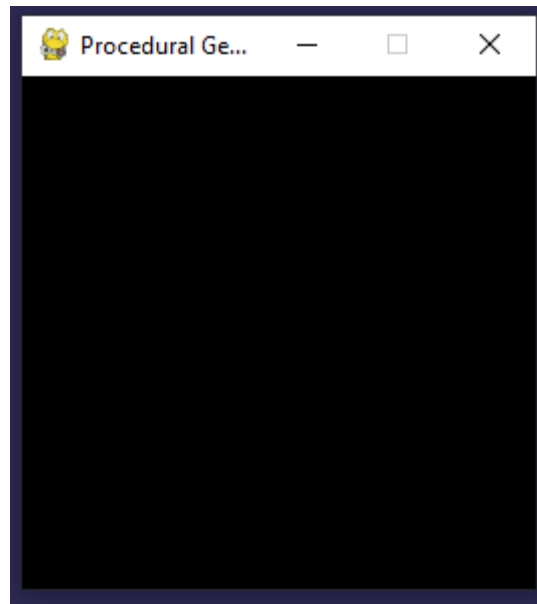
Starting from the beginning of my hierarchy I installed Pygame using *pip* and started creating a window. This was a relatively simple task only taking a few lines:

```

1  import pygame
2
3  simSize = 128
4  gridSize = 2
5
6  window = pygame.display.set_mode((simSize*gridSize, simSize*gridSize))
7  pygame.display.set_caption("Procedural Generation")
8
9  running = True
10 while running == True:
11     for event in pygame.event.get():
12         if event.type == pygame.QUIT:
13             running = False

```

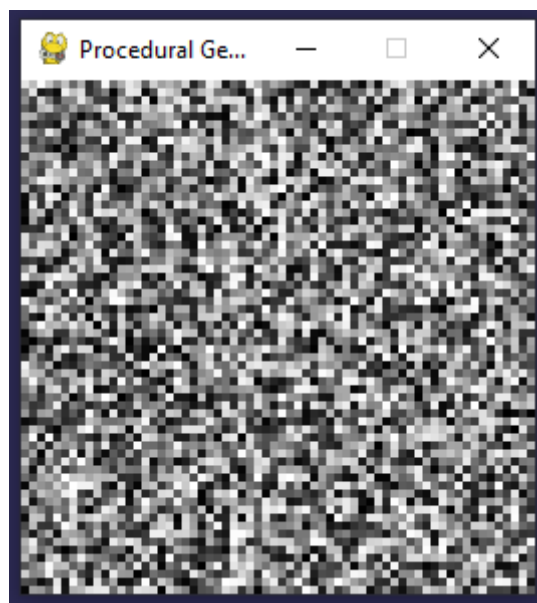
This creates a window like this:



Following the hierarchy I then added noise generation by generating random numbers and assigning them to a 2d List. Shown here:

```
1 def GenerateMap(self, seed):  
2     random.seed(seed)  
3     for y in range(0, self.arraySize):  
4         for x in range(0, self.arraySize):  
5             self.heightArray[x][y] = round(random.random(),2)
```

After creating some code to draw squares based upon the random value, I ended up with this random array of Black-White squares:



This was a good start, but didnt really look like terrain yet. As part of my research I came across simple algorithms to turn random noise into usable 2d terrain. I decided to implement these algorithms. They are relatively short and didnt take too much time to implement. I've named the two algorithms UpDownNeutralGen and Average.

UpDownNeutralGen Method

The UpDownNeutralGen method takes a tile, and considers every tile around it. It sums the tile which are greater than, less than, or within a certain range of the tile height. And then pulls the selected tile in the direction which has the highest precedence. As an example, here are some randomly generated values:

0.71	0.19	0.3
0.46	0.26	0.82
0.63	0.35	0.05

If we count the surrounding values into corresponding Higher, Lower and Neutral we get:

Higher	Lower	Neutral
4	1	3

This leads us to calculating the *pullValue*, respectively for each case:

$$\text{pullValue} = \begin{cases} \text{upTiles} \times 0.09 & \text{Most Up Tiles} \\ \text{downTiles} \times -0.08 & \text{Most Down Tiles} \\ 0 & \text{Most Neutral Tiles} \end{cases}$$

$$\text{Value}[x][y] = \text{pullValue}$$

We then add the pullValue to the original square value, leaving us with the updated value. The code for this is shown below:

```

1  def UpNeutralDownGen(self):
2      dupMap = self.heightArray
3      for y in range(0, self.arraySize):
4          for x in range(0, self.arraySize):
5              up = 0
6              down = 0
7              neutral = 0
8              pointArr = []
9
10             if x != 0 and y != 0:
11                 pointArr.append(self.heightArray[x - 1][y - 1])
12             if x != 0 and y != self.arraySize - 1:
13                 pointArr.append(self.heightArray[x - 1][y + 1])
14             if x != self.arraySize - 1 and y != self.arraySize - 1:
15                 pointArr.append(self.heightArray[x + 1][y + 1])
16             if x != self.arraySize - 1 and y != 0:
17                 pointArr.append(self.heightArray[x + 1][y - 1])
18             if x != 0:
19                 pointArr.append(self.heightArray[x - 1][y])
20             if y != 0:
21                 pointArr.append(self.heightArray[x][y - 1])
22             if x != self.arraySize - 1:
23                 pointArr.append(self.heightArray[x + 1][y])
24             if y != self.arraySize - 1:
25                 pointArr.append(self.heightArray[x][y + 1])
26

```



```

27         for i in range(len(pointArr)):
28             if pointArr[i] >= self.heightArray[x][y] + 0.1:
29                 up += 1
30             elif pointArr[i] <= self.heightArray[x][y] - 0.1:
31                 down += 1
32             else:
33                 neutral += 1
34
35         if (up > down) and (up > neutral): # Up
36             value = 0.09 * up
37         elif (down > up) and (down > neutral): # Down
38             value = -0.08 * down
39         else: # Neutral
40             value = 0
41
42         dupMap[x][y] += value
43         dupMap[x][y] = self.Clamp(dupMap[x][y], 0, 1)
44
45     self.heightArray = dupMap

```

Average Method

The Average method takes a tile and considers every tile around it, this time instead of looking at the differences, it creates an average from the 8 surrounding tiles. It then sets the selected tile to this average value. As an example, here are some randomly generated values:

0.83	0.93	0.64
0.07	0.38	0.21
0.33	0.94	0.95

Summing these and dividing by the total ammount of tiles grants us the average:

$$\frac{0.83 + 0.93 + 0.64 + 0.07 + 0.38 + 0.21 + 0.95 + 0.33 + 0.94}{9} = 0.586$$

$$Value[x][y] = 0.586$$

The code for this is shown below:

```

1  def AverageGen(self):
2      dupMap = self.heightArray
3      for y in range(0, self.arraySize):
4          for x in range(0, self.arraySize):
5              total = 0
6              count = 0
7              if x != 0 and y != 0:
8                  total += self.heightArray[x - 1][y - 1]
9                  count += 1
10             if x != 0 and y != self.arraySize - 1:
11                 total += self.heightArray[x - 1][y + 1]
12                 count += 1
13             if x != self.arraySize - 1 and y != self.arraySize - 1:
14                 total += self.heightArray[x + 1][y + 1]
15                 count += 1
16             if x != self.arraySize - 1 and y != 0:
17                 total += self.heightArray[x + 1][y - 1]

```

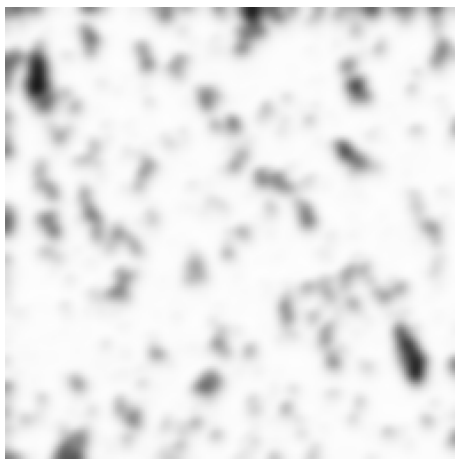
```

18         count += 1
19     if x != 0:
20         total += self.heightArray[x - 1][y]
21         count += 1
22     if y != 0:
23         total += self.heightArray[x][y - 1]
24         count += 1
25     if x != self.arraySize - 1:
26         total += self.heightArray[x + 1][y]
27         count += 1
28     if y != self.arraySize - 1:
29         total += self.heightArray[x][y + 1]
30         count += 1
31
32     dupMap[x][y] = total / count
33     self.heightArray = dupMap

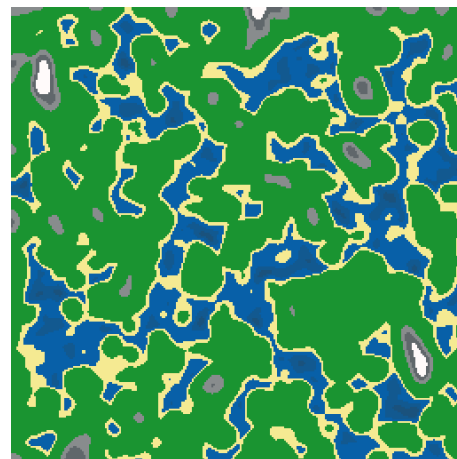
```

1.5.3 Finished Terrain Generation

Overall I am happy with the Terrain generation, though I feel as if it could be improved to look more realistic. The difference between the original random noise and the Colour Mapped Terrain looks so much better.



(a) Grayscale Terrain Generation



(b) Colour Bands applied to the Terrain Generation

1.5.4 Matrix Data Structure

As part of my Matrix Class I made a list of operations which would be key to a Matrix Class, along with being useful for Machine Learning. A Matrix is an abstract data type, commonly used in Maths, but has practical uses in the world of Computer Science. It holds a 2d array of values such as:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \begin{bmatrix} a & b & c & d \\ e & f & g & h \end{bmatrix}$$

The values in a Matrix can be manipulated using common operations such as $+$ $-$ $*$ as long as the orders of the 2 Matrices match up. Along with other, non-standard operations which have other purposes.

As part of my Matrix Class, I implemented the following operators:

1. Addition/Subtraction

Implementing Addition didnt take too long, I utilised a nested for loop to iterate over every value in both Matrices. Adding the two values together into a temporary Matrix which the method then returned.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix}$$

The written code is shown below:

```

1  @staticmethod
2  def MatrixAddSubtract(m1, m2, subtract = False):
3      m1Dims = m1.Dimensions()
4      newMat = Matrix(m1Dims[0], m1Dims[1])
5      for y in range(0, m1Dims[0]):
6          for x in range(0, m1Dims[1]):
7              if subtract:
8                  newMat.matrixArr[y][x] = m1.Val()[y][x] - m2.Val()[y][x]
9              else:
10                 newMat.matrixArr[y][x] = m1.Val()[y][x] + m2.Val()[y][x]
11     return newMat

```

2. Multiplication

Multiplication of Matrices is slightly more complicated, it is of $O(n^3)$ complexity, utilising a triple nested for loop. It multiplies the row of a $M1$, by the column in $M2$. Summing the calculation into the element in the new Matrix $M3$.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a*e + b*g & a*f + b*h \\ c*e + d*g & c*f + d*h \end{bmatrix}$$

There is also Scalar Multiplication which multiplies each value of a Matrix by the Scalar.

$$k * \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} ka & kb \\ kc & kd \end{bmatrix}$$

The written code is shown below:

```

1  @staticmethod
2  def ScalarMultiply(s, m1):
3      m1Dims = m1.Dimensions()
4      newMat = Matrix(m1Dims[0], m2Dims[1])
5      for y in range(0, m1Dims[0]):
6          for x in range(0, m1Dims[1]):
7              newMat.matrixArr[y][x] = m1.matrixArr[y][x] * s
8
9  @staticmethod
10 def MatrixMultiply(m1, m2):
11     m1Dims = m1.Dimensions()
12     m2Dims = m2.Dimensions()
13     newMat = Matrix(m1Dims[0], m2Dims[1])
14     for row in range(0, m1Dims[1]):
15         subRow = m1.Val()[row][0:m1Dims[1]]
16         for col in range(0, m2Dims[1]):
17             subCol = []
18             for i in range(0, m1Dims[0]):
19                 print(i)
20                 subCol.append(m2.Val()[i][col])
21             total = 0
22             for x in range(0, len(subRow)):

```

```

23         total += subRow[x] * subCol[x]
24         newMat.matrixArr[row][col] = total
25     return newMat

```

3. Determinant

Calculating the Determinant of an NxN Matrix is a recursive algorithm. With the base case being the Determinant of a 2x2 Matrix. When calculating the Determinant of a 3x3 Matrix you create a Matrix of Cofactors, and multiply each value by the corresponding value in the Sin Matrix (*Formed from repeating 1's and -1's*). Summing the values from a singular Row or Column will then give you the Determinant. For a 4x4 you simply calculate the Determinant of the corresponding 3x3's to get the Cofactors.

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = a * d - b * c$$

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a * \begin{vmatrix} e & f \\ h & i \end{vmatrix} - b * \begin{vmatrix} d & f \\ g & i \end{vmatrix} + c * \begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

The written code is shown below:

```

1  def SubMatrixList(self, rowList, colList):
2      newMat = Matrix(self.Dimensions()[0] - len(rowList), self.Dimensions()[1] - len(colList))
3      xoffset = 0
4      yoffset = 0
5      yRowList = []
6
7      for y in range(0, self.Dimensions()[0]):
8          for x in range(0, self.Dimensions()[1]):
9              if x in colList and y in rowList:
10                 xoffset += 1
11                 yoffset += 1
12                 continue
13             elif x in colList:
14                 xoffset += 1
15                 continue
16             elif y in rowList and y not in yRowList:
17                 yoffset += 1
18                 yRowList.append(y)
19                 continue
20             else:
21                 newMat.matrixArr[y - yoffset][x - xoffset] = self.matrixArr[y][x]
22             xoffset = 0
23         return newMat
24
25     @staticmethod
26     def Determinant(m):
27         dims = m.Dimensions()
28         if dims[1] <= 2:
29             det = (m.matrixArr[0][0] * m.matrixArr[1][1]) - (m.matrixArr[0][1] * m.matrixArr[1][0])
30             return (det)
31         elif dims[1] != 2:
32             det = 0
33             subtract = False
34             tempMat = m.SubMatrixList([0], [])
35             for i in range(0, dims[1]):
36                 subMat = None
37                 subMat = m.SubMatrixList([0], [i])
38                 if subtract == False:

```

```

39         det += m.matrixArr[0][i] * Matrix.Determinant(subMat)
40         subtract = True
41     elif subtract == True:
42         det -= m.matrixArr[0][i] * Matrix.Determinant(subMat)
43         subtract = False
44     return det

```

4. Dot Product

The Dot Product occurs between two vectors, and can be used to calculate the angle between them. Its a relatively simple operation only taking a few lines of code.

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \cdot \begin{bmatrix} d \\ e \\ f \end{bmatrix} = a \times d + b \times e + c \times f$$

The written code is shown below:

```

1  @staticmethod
2  def DotProduct(v1,v2):
3      total = 0
4      for i in range(v1.Dimensions()[0]):
5          total += v1.Val()[i][0] * v2.Val()[i][0]
6      return total

```

1.5.5 Prototype Evaluation

Overall I am happy with my prototype, though I feel like some parts need to be improved. I did meet my objectives for my prototype but there were improvements which can be made when I create my Technical Solution. Namely the Terrain Generation along with the Matrix class. I feel that Perlin noise would be a better alternative to the two algorithms I used. In theory it should produce better results, and also provide more marks for complexity. My Matrix class could be rewritten to be more efficient, along with using operator overloading, which I didn't know Python could do at the time. I also feel like having Vector inherit from Matrix is relatively pointless, there is no need for it when I could just use 1 wide Matrices instead.

1.6 Second Interview

I asked a few more questions to my Expert regarding my project at this point. Receiving feedback on my Prototype and gaining a greater understanding of the Machine Learning Model I'm intending to use.

1. What are your thoughts on my prototype?

"I think your prototype is good, but could be improved. The use of Operator Overloading would improve your Matrix Class, and optimising some of your algorithms would be useful. The Terrain generation looks good, but I think it's a bit water heavy, this is where Perlin Noise might help you to achieve better results. Would also be more fine tunable to your liking."

2. Is a Dual Neural Network a good model to choose?

"A Dual Neural Network should in theory be a complex enough Model for your project. The concern I have is whether your Network will be able to generalise enough in order to sufficiently 'solve' the simulation you design. There are some algorithms you could implement in order to tackle this though. You could do some research into these before finalising your design."

3. Which Activation Functions should I implement?

"The most commonly used are Sigmoid, TanH, ReLu and SoftMax. They are relatively simple so wont take long to implement. Those would be a good starting point for testing your Neural Network."

4. What type of Reward system should I use?

"As far as I'm aware there are two types of reward systems, Sparse and Dense. I think that Sparse would be better suited to your project. Sparse is where the reward given to the Agent is 0 for most actions. Compared to dense where reward is given for most most actions."

1.7 Objectives

Taking into account my Prototype and Interview, I have formed a list of objectives I feel to be most appropriate for my Investigation.

If all completed they will form a complete solution which will answer my Investigations question. Below is the list of objectives split into 6 key sections:

User Input

1. Read Parameters from a Json formatted file
2. Check Parameters fall within a certain range to prevent errors
3. Give user option to load Neural Network Training progress

Simulation

1. Utilise Perlin Noise to generate a 2d List of terrain heights
2. Store Terrain Heights in a Tile Data Type
3. Utilise Threading to generate Terrain Faster
4. Display terrain to a window
5. Map ranges of terrain heights to specific colour bands
6. Utilise Poisson Disc Sampling to generate objects for the Agent to interact with
7. Implement enemies which use basic pathfinding to traverse towards the player
8. Generate multiple enemies upon starting the simulation
9. Allow the enemies to attack the Agent

Agent

1. Implement Movement options for the Agent
2. Implement the ability to pick up the generated Objects
3. Implement the ability to attack the generated enemies
4. Create methods to sample the terrain around the Agent
5. Create methods to convert the sampled Tiles into a grayscale input vector for a neural network
6. Create reward methods to reward the agent given the terrain samples and action

Matrix Class

1. Implement a Dynamic Matrix Class with appropriate Operations such as:
 - Multiplication
 - Addition
 - Subtraction
 - Transpose
 - Sum
 - Select Row/Column
2. Create appropriate errors to throw when utilising methods the incorrect way

Deep Reinforcement Learning

1. Dynamically create a Dual Neural Network model based upon loaded parameters
2. Implement an Abstract Class for Activation Functions
3. Implement Activation Functions inheriting from the Abstract Class such as:
 - Sigmoid
 - TanH
 - ReLu
 - Leaky ReLu
 - SoftMax
4. Create methods to Forward Propagate the neural network
5. Create methods to calculate the loss of the network using the Bellman Equation
6. Create methods to Back Propagate calculated error through the neural network
7. Create methods to update weights and biases within the network to converge on a well trained network
8. Utilise the outlined Matrix class to perform the mathematical operations in the specified methods
9. Implement Load and Save Methods to save progress in training
10. Implement a Double Ended Queue/Deque Data Type
11. Implement Experience Replay utilising the Deque Data Type to increase training accuracy

Data Logger

1. Be able to create a Data Logger class to log data points across training
2. Be able to create a Data Structure for the Data Logger
3. Allow multiple types specified types for a single parameter
4. When adding a new Data Point the Logger will check it to make sure it matches the given Data Structure
5. Implement a Heap Data Type
6. Implement a Heap sort using the Heap Data Type
7. Be able to sort by a parameter in the Data Structure
8. Be able to select a single parameter from the data points
9. Implement Load and Save Functions to save progress during training

1.8 Modelling of the Problem

In this section I will define and derive all the Mathematical Formulae relating to my Project. This includes all the Matrix Operations I plan to use and the General Forms of Forward Propagation and Back Propagation.

1.8.1 Matrices

Overview

Matrices are a Mathematical Data Structure, storing elements in the shape of a Rectangle. They are arranged Rows and Columns. An $m \times n$ Matrix will have m Rows and n Columns. As part of defining the Matrix Operations, below is defined Matrix A and Matrix B and can be of any size.

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,m} \\ a_{2,1} & a_{2,2} & \dots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,m} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,m} \\ b_{2,1} & b_{2,2} & \dots & b_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,1} & b_{n,2} & \dots & b_{n,m} \end{bmatrix}$$

Matrix Addition

Matrix Addition is the Operation of adding two Matrices by adding the Corresponding Elements together. Matrix Addition is Commutative. Below is A added to B .

$$A + B = \begin{bmatrix} a_{1,1} + b_{1,1} & a_{1,2} + b_{1,2} & \dots & a_{1,m} + b_{1,m} \\ a_{2,1} + b_{2,1} & a_{2,2} + b_{2,2} & \dots & a_{2,m} + b_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} + b_{n,1} & a_{n,2} + b_{n,2} & \dots & a_{n,m} + b_{n,m} \end{bmatrix}$$

Matrix Subtraction

Matrix Subtraction is the Operation of subtracting two Matrices by adding the Corresponding Elements together, with the 2nd Matrix's element being Negated. Below is B Subtracted from A .

$$A - B = \begin{bmatrix} a_{1,1} - b_{1,1} & a_{1,2} - b_{1,2} & \dots & a_{1,m} - b_{1,m} \\ a_{2,1} - b_{2,1} & a_{2,2} - b_{2,2} & \dots & a_{2,m} - b_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} - b_{n,1} & a_{n,2} - b_{n,2} & \dots & a_{n,m} - b_{n,m} \end{bmatrix}$$

Matrix Multiplication

Matrix Multiplication calculates the Dot Product between the Rows in Matrix A and Columns in Matrix B . The Dot Product is a Vector Operation which takes two equal-length series of Numbers and returns a single Number. Each element in the 1st series of Numbers is Multiplied with the opposing element in the 2nd series, these are then summed to find the Dot Product.

$$AB = \begin{bmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,m} \\ c_{2,1} & c_{2,2} & \dots & c_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n,1} & c_{n,2} & \dots & c_{n,m} \end{bmatrix}$$

Such that

$$c_{i,j} = a_{i,1}b_{1,j} + a_{i,2}b_{2,j} + \dots + a_{i,n}b_{n,j} = \sum_{k=1}^n a_{i,k}b_{k,j}$$

Matrix Scalar Multiplication

Scalar Multiplication Multiplies each element by a single Scalar, in this case k .

$$k * A = \begin{bmatrix} ka_{1,1} & ka_{1,2} & \dots & ka_{1,m} \\ ka_{2,1} & ka_{2,2} & \dots & ka_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ ka_{n,1} & ka_{n,2} & \dots & ka_{n,m} \end{bmatrix}$$

Matrix Hadamard Product

The Hadamard Product calculates the element-wise product between two equally sized Matrices.

$$A \odot B = \begin{bmatrix} a_{1,1}b_{1,1} & a_{1,2}b_{1,2} & \dots & a_{1,m}b_{1,m} \\ a_{2,1}b_{2,1} & a_{2,2}b_{2,2} & \dots & a_{2,m}b_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1}b_{n,1} & a_{n,2}b_{n,2} & \dots & a_{n,m}b_{n,m} \end{bmatrix}$$

Matrix Transpose

The Transpose of a Matrix flips the given Matrix over the Diagonal, effectively Rows become Columns.

$$B^T = \begin{bmatrix} b_{1,1} & b_{2,1} & \dots & b_{n,1} \\ b_{1,2} & b_{2,2} & \dots & b_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ b_{1,m} & b_{2,m} & \dots & b_{n,m} \end{bmatrix}$$

1.8.2 Forward Propagation

Overview

Forward Propagation is used in a Neural Network to calculate the output of the Network. It feeds Input Data through each Layer, leaving each Node with its resultant Activation Value. This is completed in two processes: Pre-Activation and Activation.

The Standard Notation I will be using to describe the Calculations:

$a_i^{(L)}$ = The Activation Value for the i^{th} Node in the L^{th} Layer

$z_i^{(L)}$ = The Pre-Activation Value for the i^{th} Node in the L^{th} Layer

$w_{m,n}^{(L)}$ = The Weight between node $n \rightarrow m$ from the L^{th} to the $(L+1)^{th}$

$b_i^{(L)}$ = The Bias Value for the i^{th} Node in the L^{th} Layer

Pre-Activation

The Pre-Activation Value for the i^{th} Node is the Sum of the Preceding Layers Activation Values, Multiplied by the Weight value between them. This then has the Bias Value added. M is the size the Layer $(L-1)$.

$$z_i^{(L)} = \sum_{k=1}^M (a_i^{(L-1)} \times w_{k,i}^{(L-1)}) + b_i^{(L)}$$

This can also be represented in its Matrix Form rather easily. You take the Vector of Activation Values from $(L - 1)$ and multiply it by the Weight Matrix from $(L - 1)$. You then add the Vector of Bias Values and that leaves you with the Pre-Activation for Layer L .

$$Z^{(L)} = W^{(L-1)} \times A^{(L-1)} + B^{(L)}$$

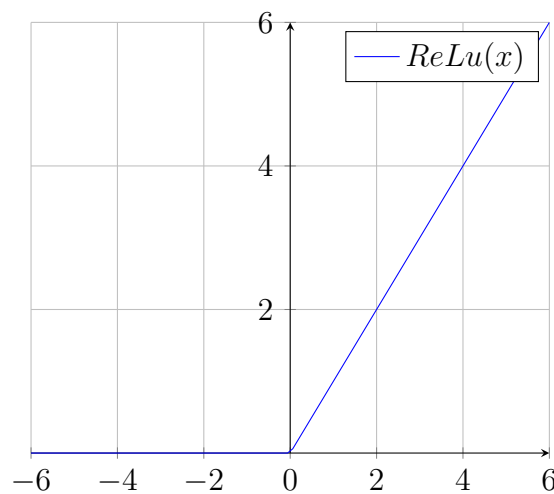
$$Z^{(L)} = \begin{bmatrix} z_0^{(L)} \\ z_1^{(L)} \\ \vdots \\ z_n^{(L)} \end{bmatrix} = \begin{bmatrix} w_{0,0}^{(L-1)} & w_{0,1}^{(L-1)} & \dots & w_{0,m}^{(L-1)} \\ w_{1,0}^{(L-1)} & w_{1,1}^{(L-1)} & \dots & w_{1,m}^{(L-1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,0}^{(L-1)} & w_{n,1}^{(L-1)} & \dots & w_{n,m}^{(L-1)} \end{bmatrix} \begin{bmatrix} a_0^{(L-1)} \\ a_1^{(L-1)} \\ \vdots \\ a_n^{(L-1)} \end{bmatrix} + \begin{bmatrix} b_0^{(L)} \\ b_1^{(L)} \\ \vdots \\ b_n^{(L)} \end{bmatrix}$$

Activation

Activation Functions are usually an abstraction representing the rate of "Action Potential" firing in the Node. The most Common Activations for Neural Networks are the following 4 Activations:

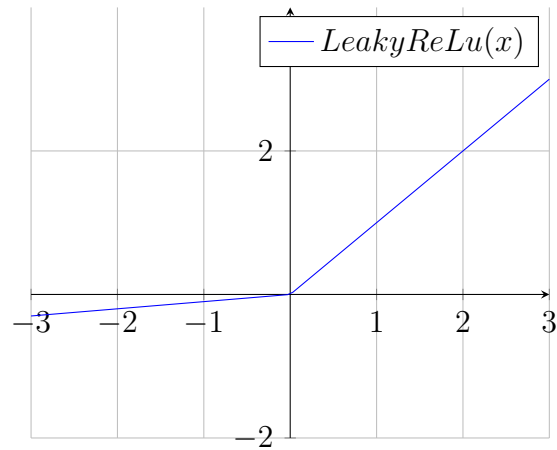
ReLU

$$\text{ReLU}(x) = \begin{cases} x & x < 0 \\ 0 & x > 0 \end{cases}$$

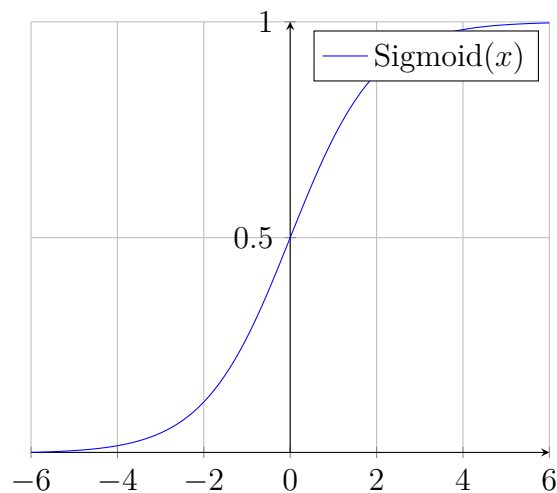


Leaky ReLu

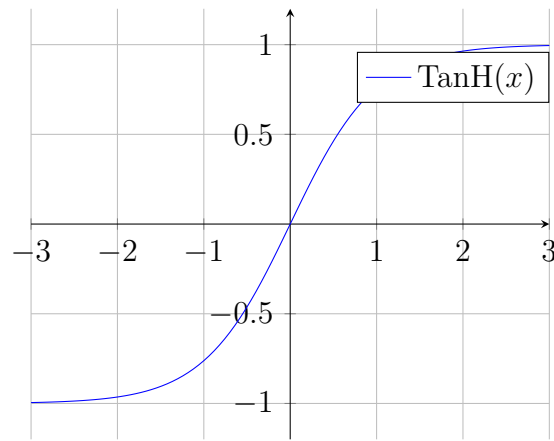
$$\text{ReLU}(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$$

**Sigmoid**

$$\text{Sigmoid}(x) = \frac{1}{1+e^{-x}}$$

**TanH**

$$\text{TanH}(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



SoftMax

SoftMax is an exception to the Activation Functions and is a Generalisation of Sigmoid to Multiple Dimensions. It takes in a Vector \mathbf{z} of K Real Numbers, and normalises it into a probability distribution which Sums to 1.

$$\text{SoftMax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

For $i = 1, \dots, K$ And $\mathbf{z} = (z_1, \dots, z_K)$

1.8.3 Differentiation

Differentiation from First Principles

Differentiation is the process of finding the Gradient of a Function at a specific point. In the case of a Neural Network, this can be used to measure the sensitivity of the Function Output, in respect to the Input. This derivative is known as said Functions' Gradient Function.

With a simple straight line graph we can find the gradient as $\frac{\Delta y}{\Delta x}$, Δ (Delta) is used to represent a finite increment.

When find the Derivative of a more Complex Function we can use Two Points. Point $P : (x, f(x))$ and Point $Q : (x + h, f(x + h))$. The variable h tends towards 0, so Points Q will eventually be ontop of point P . This is called Differentiation from First Principles.

$$\begin{aligned} \frac{\Delta y}{\Delta x} &= \lim_{h \rightarrow 0} \left(\frac{f(x + h) - f(x)}{(x + h) - x} \right) \\ &= \lim_{h \rightarrow 0} \left(\frac{f(x + h) - f(x)}{h} \right) \end{aligned}$$

Derivatives are more commonly represented as $f'(x)$ or $\frac{dy}{dx}$

Standard Differentiation Rules

Instead of manually using Smaller and Smaller Values of h manually, there are standard Differentiation Rules. These are as follows:

$$\begin{aligned}
y = x^k &\rightarrow \frac{dy}{dx} = kx^{k-1} \\
y = k &\rightarrow \frac{dy}{dx} = 0 \\
y = e^{kx} &\rightarrow \frac{dy}{dx} = ke^{kx} \\
y = f(x)g(x) &\rightarrow f(x)g'(x) + f'(x)g(x) \\
f(x) = \frac{g(x)}{h(x)} &\rightarrow f'(x) = \frac{g'(x)h(x) - g(x)h'(x)}{h(x)^2}
\end{aligned}$$

These rules are applied to each component of the Function to find the Derivative.

Chain Rule

The Chain Rule is used to compute the derivative of Nested Functions such as $f(x) = g(h(x))$. The derivative of this Function can be expressed as:

$$f'(x) = g'(h(x))h'(x)$$

This can be applied to an infinite number of Functions, where $f(x) = g_1(g_2(\dots(g_n(x))))$. By this rule we can represent the derivative as a Series of Derivatives Multiplied together:

$$\frac{df}{dx} = \frac{df}{df_1} \frac{df_1}{df_2} \frac{df_2}{df_3} \dots \frac{df_n}{df_x}$$

Partial Derivatives

Partial Derivatives are used when the Function in question contains Multiple Variables. They utilise the same rules, except the Variables which aren't being derived get treated as constants. The Derivative of $f(x, y)$ with respect to x is expressed as $f'_x(x, y)$ or $\frac{\partial f}{\partial x}$.

1.8.4 Back Propagation

Overview

Back Propagation is the algorithm used to adjust Weights and Bias' in a Neural Network. Through using this algorithm you can successfully "train" the Network to recognise certain patterns in data. The Input Data gets propagated through the Network using Forward Propagation, and then the output is passed into the Loss Function.

The Bellman Equation

The Bellman Equation is a method of optimisation, and is used for dynamic programming. In the context of Machine Learning we can utilise it to reinforce good behaviour and negate bad behaviour. By writing the relationships between two states in the form of an action, we can optimise this by choosing the best action when given a state. If we let s_t be the current state, we can define all the possible actions from that state as $a_t \in \Gamma(s_t)$. Where $\Gamma(s_t)$ represents all given actions from a state. We can also define the State Transition from $s_t \rightarrow s_{t+1}$ as $T(s_t, a)$ when action a has been taken. The Reward from this is given as $R(s_t, a)$. A Discount Factor

$0 < \gamma < 1$ is also defined to assume impatience, compounding the effects of γ the further in the future the Reward is.

With these definitions, an infinite-horizon problem is formed:

$$V(s_0) = \max_{\{a_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \gamma^t \cdot R(s_t, a_t)$$

We can form this into another Equation which uses the Principle of Optimality, such that:

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. - Richard E. Bellman

We will consider the first decision separately to all future reward, and then collect the future decisions within the brackets, which the infinite-horizon problem above is equivalent too.

$$\max_{a_0} \left\{ R(s_0, a_0) + \gamma \cdot \left[\max_{\{a_t\}_{t=1}^{\infty}} \sum_{t=1}^{\infty} \gamma^{t-1} \cdot R(s_t, a_t) \right] \right\}$$

This at first glance has only made the problem uglier but infact has made our lives easier. It can be condensed further into a Recursively Defined Function:

$$V(s_0) = \max_{a_0} \{ R(s_0, a_0) + \gamma \cdot V(x_1) \}$$

When subjected to: $x_1 = T(s_0, a_0)$

Loss Function

The Loss Function of a Network represents how well a Neural Network is performing. The aim of the Back Propagation is to minimise this Functions output. When using a standard Neural Network and you're training on a labelled data set, you can be certain about the Expected Output. The standard Loss Function is as follows:

$$\begin{aligned} Loss_i &= \frac{1}{2} \cdot (ExpectedOutput_i - ActualOutput_i)^2 \\ &= \frac{1}{2} \cdot (y_i - \hat{y}_i)^2 \end{aligned}$$

This is whats called the Half Square Difference. This Differentiates nicely which is why it is commonly used.

We use the Bellman Equation to calculate the expected value for the loss function.:

$$\begin{aligned} Q(s_t, a_t) &= R(s_t, a_t) + \gamma \cdot \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \\ y &= \left(R(s_t, a_t) + \gamma \cdot \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right)^2 \end{aligned}$$

Gradient Descent

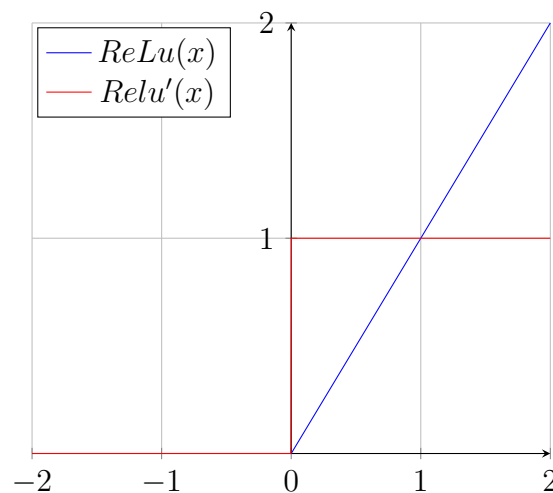
To minimise the Loss Function, the Weights and Bias' in the Network need to be algorithmically adjusted to converge towards the expected outputs. You can calculate these adjustments by using Partial Derivatives. You can take the Derivative of every Weight and Bias with respect to the Loss Function. The Derivatives of each weight can vary, such as one weight being 0.5 and the other being 3, the Second Weight affects the Loss Function 10× as much. This process is known as Gradient Descent.

Differentiating Activation Functions

As part of Back Propagation we need to derive all the Activation Functions we use within our Layer structure. The Derivatives are shown below.

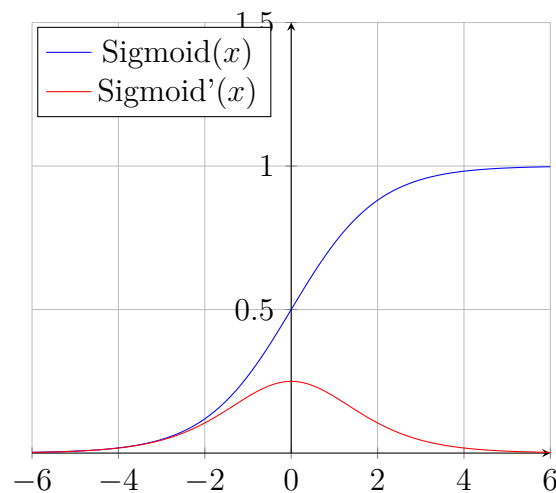
The ReLu Derivative:

$$\text{ReLu}(x) = \begin{cases} 0 & x < 0 \\ x & x > 0 \end{cases}$$
$$\text{Relu}'(x) = \begin{cases} 0 & x < 0 \\ 1 & x > 0 \end{cases}$$



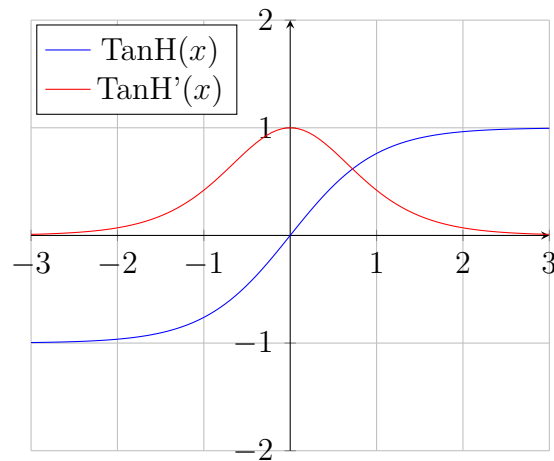
The Sigmoid Function Derivative:

$$\begin{aligned}
 \text{Sigmoid}(x) &= \frac{1}{1+e^{-x}} \\
 &= (1+e^{-x})^{-1} \\
 \frac{d\sigma(x)}{dx} &= -1 \cdot (1+e^{-x})^{-2} \cdot -e^{-x} \\
 &= \frac{e^{-x}}{(1+e^{-x})^2} \\
 &= \frac{e^{-x}}{1+e^{-x}} \cdot \frac{1}{1+e^{-x}} \\
 &= \frac{e^{-x}+1-1}{1+e^{-x}} \cdot \frac{1}{1+e^{-x}} \\
 &= \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}} \right) \cdot \frac{1}{1+e^{-x}} \\
 &= \text{Sigmoid}(x) \cdot (1 - \text{Sigmoid}(x))
 \end{aligned}$$



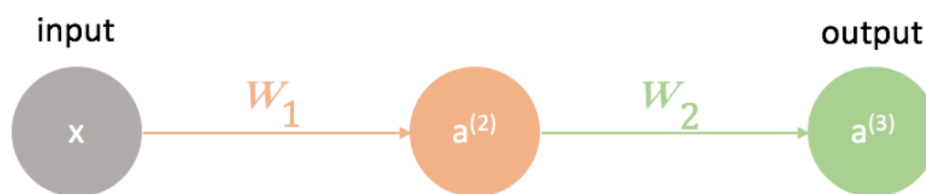
The TanH Derivative:

$$\begin{aligned}
 \text{TanH}(x) &= \frac{\sinh(x)}{\cosh(x)} \\
 &= \frac{e^x - e^{-x}}{e^x + e^{-x}} \\
 \text{TanH}'(x) &= \frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2} \\
 &= \frac{(e^x + e^{-x})^2}{(e^x + e^{-x})^2} - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2} \\
 &= 1 - \text{TanH}^2(x)
 \end{aligned}$$



Simple Network

We can apply Back Propagation to this simple Neural Network:



For this Network we need to calculate the derivative of each weight with respect to the cost function. With the use of the chain rule w_1 can be expressed as:

$$\frac{\partial c}{\partial w_2} = \frac{\partial c}{\partial a_3} \frac{\partial a_3}{\partial z_3} \frac{\partial z_3}{\partial w_2}$$

This means we need to find each derivative in the chain. The first derivative is given as $\frac{\partial c}{\partial a_3}$.

$$c = \frac{1}{2} \cdot (y - a_3)^2$$

$$\frac{\partial c}{\partial a_3} = y - a_3$$

Next we find $\frac{\partial a_3}{\partial z_3}$, here we will use TanH for our activation function.

$$a_3 = \frac{e^{z_3} - e^{-z_3}}{e^{z_3} + e^{-z_3}}$$

$$= \tanh(z_3)$$

$$\frac{\partial a_3}{\partial z_3} = 1 - \tanh^2(a_3)$$

Next we find the final derivative $\frac{\partial z_3}{\partial w_2}$

$$\begin{aligned} z_3 &= a_2 \cdot w_2 \\ \frac{\partial z_3}{\partial w_2} &= a_2 \end{aligned}$$

We then combine this all together to find $\frac{\partial c}{\partial w_2}$

$$\frac{\partial c}{\partial w_2} = (y - a_3) \cdot (1 - \tanh^2(a_3)) \cdot a_2$$

When calculating the derivatives of w_1 it's slightly more complicated, it requires us to *extend* the chain of derivatives.

$$\frac{\partial c}{\partial w_1} = \frac{\partial c}{\partial a_3} \frac{\partial a_3}{\partial z_3} \frac{\partial z_3}{\partial a_2} \frac{\partial a_2}{\partial z_2} \frac{\partial z_2}{\partial w_1}$$

It is however similar to our original chain, the only new derivative is $\frac{\partial z_3}{\partial a_2}$. Which is simply w_2 , leaving us with the following derivative:

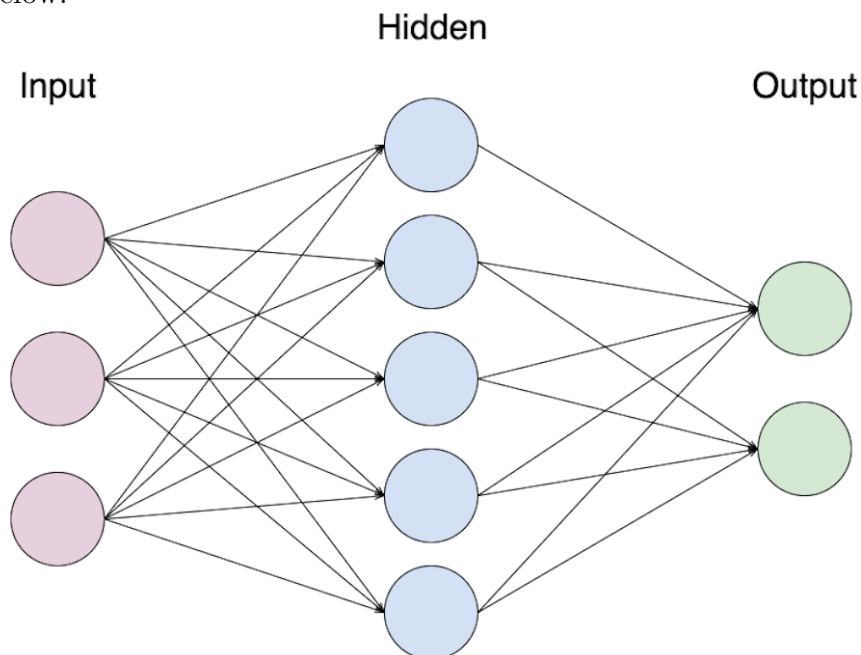
$$\frac{\partial c}{\partial w_1} = (y - a_3) \cdot (1 - \tanh^2(a_3)) \cdot w_2 \cdot a_2 \cdot (1 - \tanh^2(a_2)) \cdot a_1$$

We can generalise this into the form below for layers $1, 2, \dots, n$:

$$\begin{aligned} \frac{\partial c}{\partial w_l} &= a_l \cdot \sigma'(z_{l+1}) \cdot \frac{\partial c}{\partial a_{l+1}} \\ \frac{\partial c}{\partial a_l} &= \begin{cases} y - \hat{y} & l = n \\ w_l \cdot \sigma'(z_{l+1}) \cdot \frac{\partial c}{\partial a_{l+1}} & Else \end{cases} \end{aligned}$$

Complex Network

For a Complex Network, with multiple Neurons per layer, it is quite similar. An example of this is shown below:



When deriving an Activation value we instead need to consider all weight derivatives connected to the next layer. We can generalise this into the weight update form for layers $1, 2, \dots, n$:

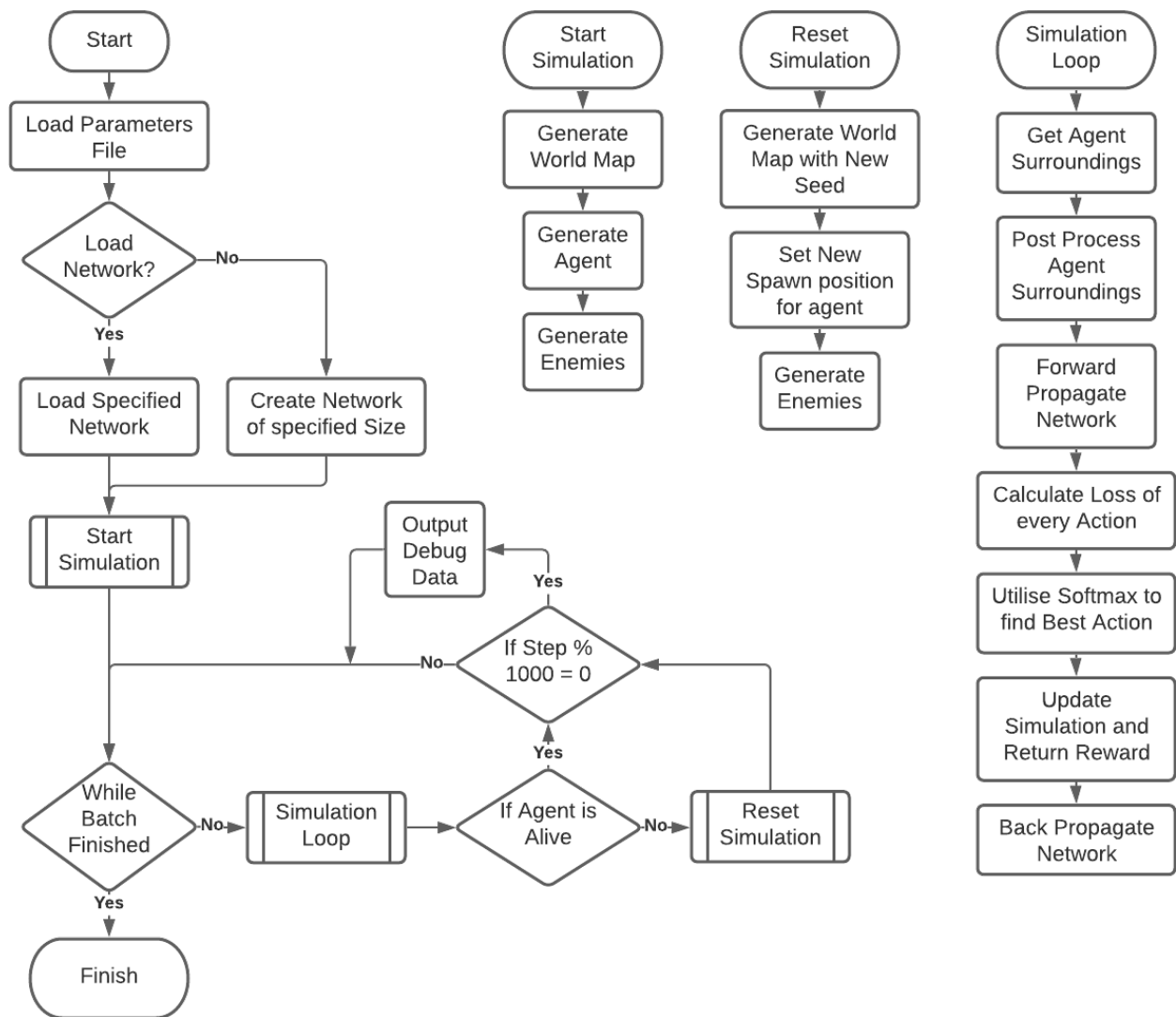
$$\begin{aligned}\Delta w_{i \rightarrow j} &= -\eta \delta_j z_i \\ \delta_i &= \begin{cases} \sigma'(z_i) \cdot (y_i - \hat{y}_i) & \text{Node } i \text{ in Final Layer} \\ \sigma'(z_i) \sum_{k \in \text{outs}(i)} \delta_k w_{i \rightarrow k} & \text{Else} \end{cases}\end{aligned}$$

This should be all that is needed to perform Back Propagation.

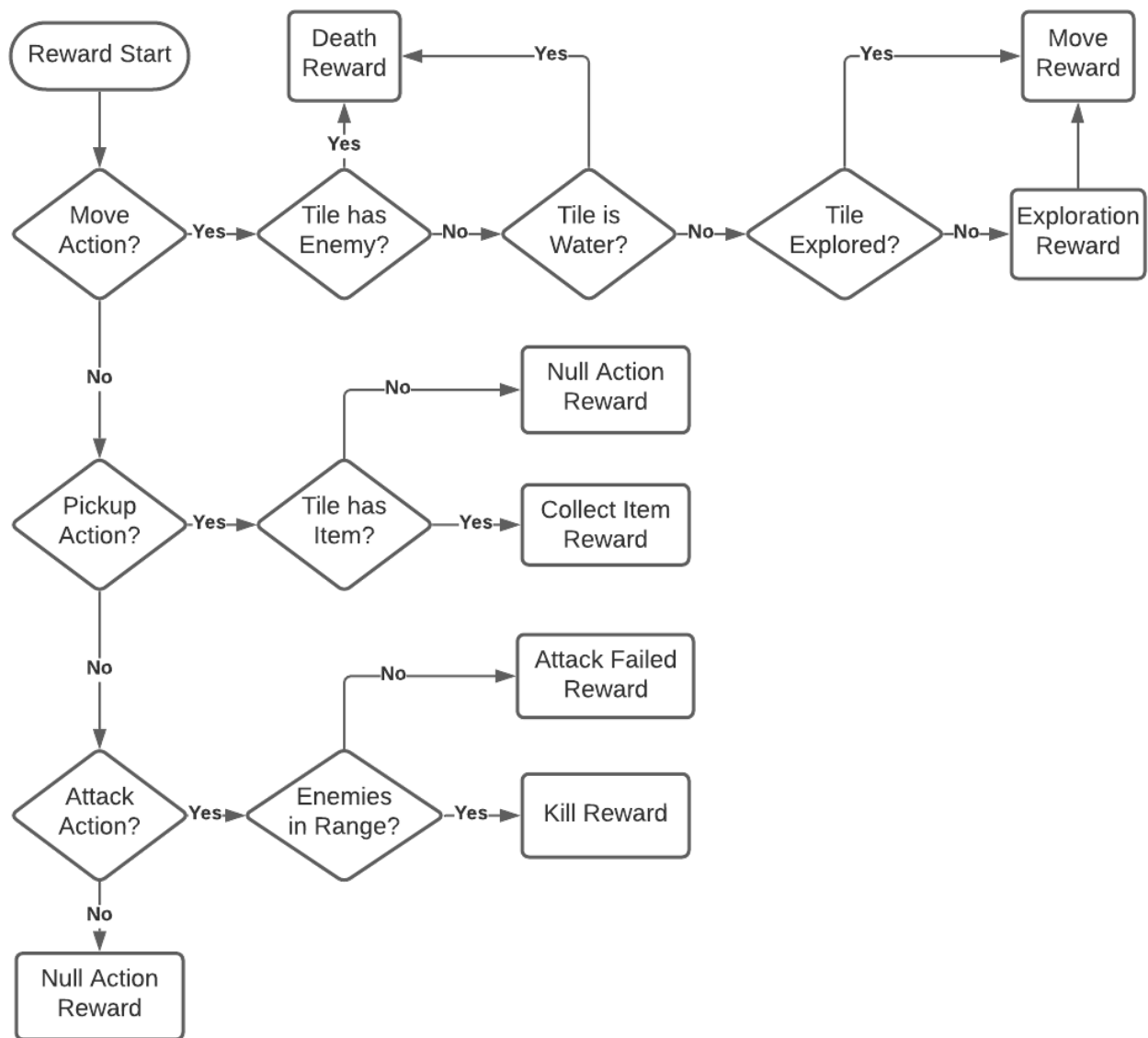
2 Design

2.1 System Flow Charts

Below is shown the Flow Chart Overview of my Entire Project. This flowchart is very abstracted without going into the fine detail of each Process.

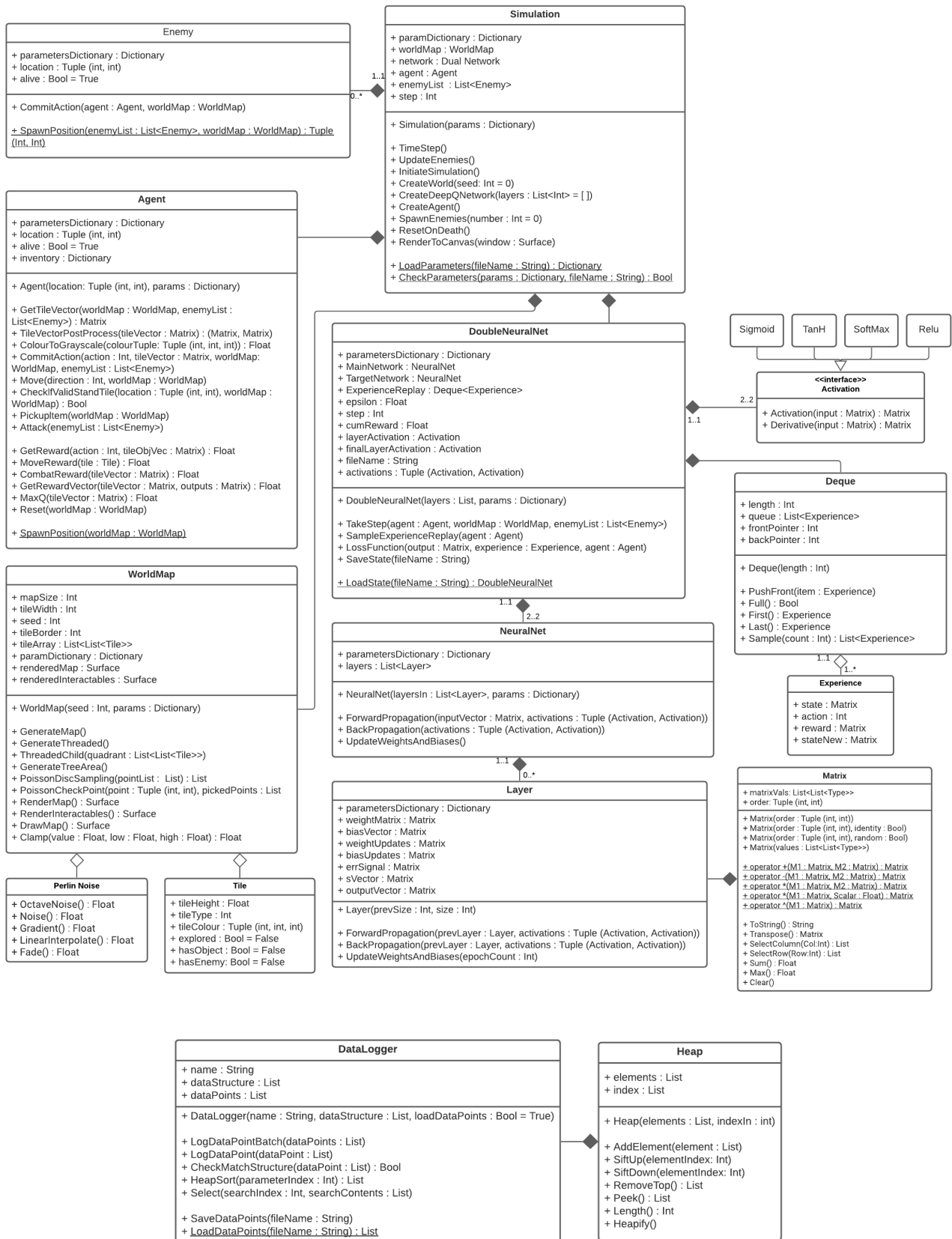


Below is shown the Action and Reward Tree for the Agent. Any Reward is added to a Total Reward Buffer and returned as part of the Function.



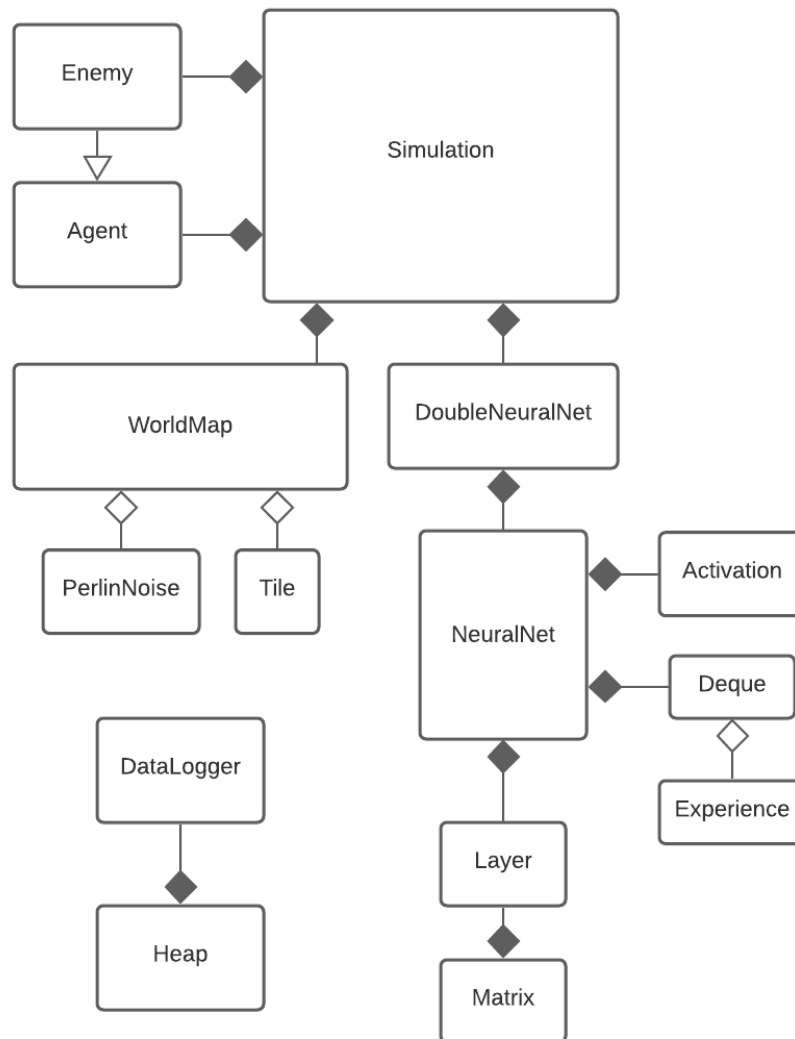
2.2 Class Diagrams

Below is shown the Class Diagram of the entire Technical Solution. The Data Logger is listed separately for clarity, as in practice multiple sections of the Program will aggregate with it.

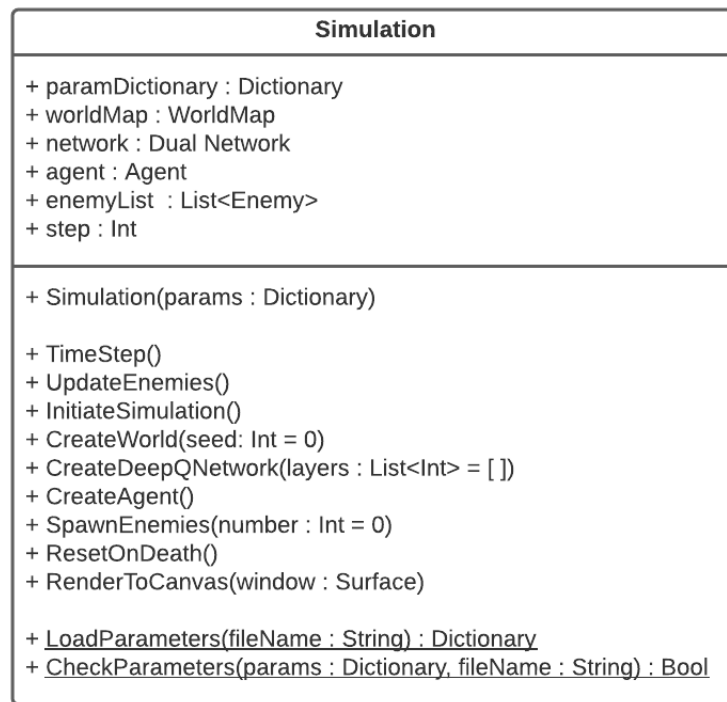


2.3 Individual Classes

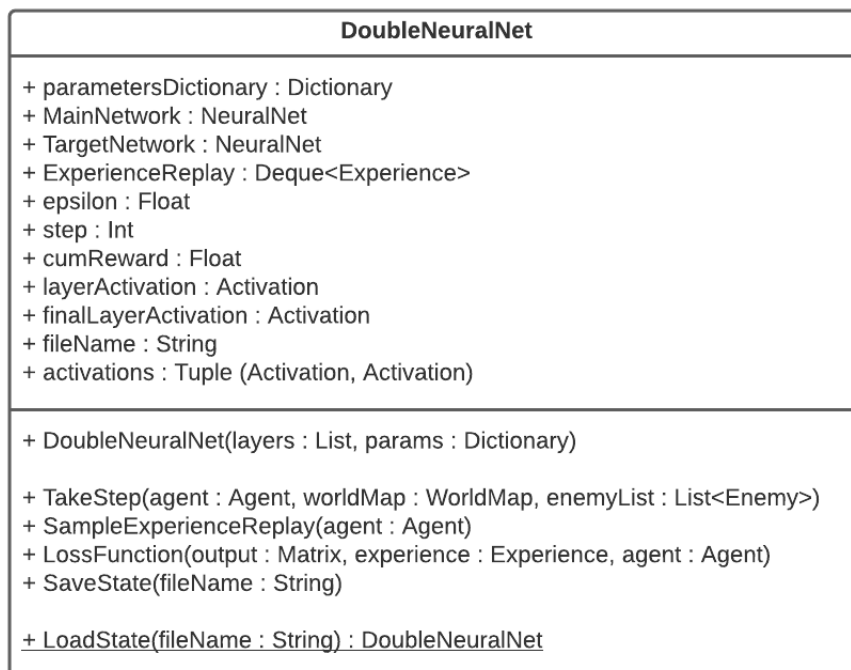
Below is shown a simplified Class Diagram, and the Individual Classes with Descriptions as to their Role in the Program.



The Simulation Class is used to compose the 3 Main Sections of the Program into a Single Interface Class, it contains all the Setup, Display and Forward Methods.



The DoubleNeuralNet Class combines together two Neural Networks to create a more complex Machine Learning Model.



The NeuralNet Class contains all the methods needed for a Functional Neural Network.

NeuralNet
+ parametersDictionary : Dictionary + layers : List<Layer>
+ NeuralNet(layersIn : List<Layer>, params : Dictionary) + ForwardPropagation(inputVector : Matrix, activations : Tuple (Activation, Activation)) + BackPropagation(activations : Tuple (Activation, Activation)) + UpdateWeightsAndBiases()

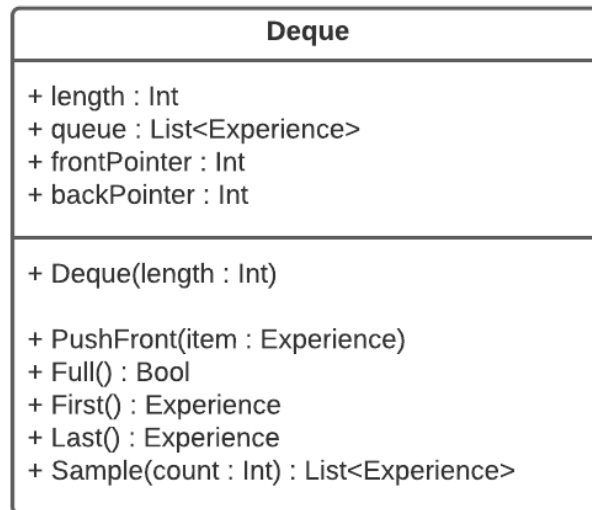
The NeuralNetwork Class contains an Array of Layer objects. They are integral to the function of the Neural Network.

Layer
+ parametersDictionary : Dictionary + weightMatrix : Matrix + biasVector : Matrix + weightUpdates : Matrix + biasUpdates : Matrix + errSignal : Matrix + sVector : Matrix + outputVector : Matrix
+ Layer(prevSize : Int, size : Int) + ForwardPropagation(prevLayer : Layer, activations : Tuple (Activation, Activation)) + BackPropagation(prevLayer : Layer, activations : Tuple (Activation, Activation)) + UpdateWeightsAndBiases(epochCount : Int)

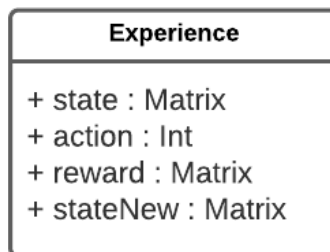
The Matrix Class is a key part of the Program, being used within the Neural Networks Logic.

Matrix
+ matrixVals: List<List<Type>> + order: Tuple (int, int)
+ Matrix(order : Tuple (int, int)) + Matrix(order : Tuple (int, int), identity : Bool) + Matrix(order : Tuple (int, int), random : Bool) + Matrix(values : List<List<Type>>)
<u>+ operator +(M1 : Matrix, M2 : Matrix) : Matrix</u> <u>+ operator -(M1 : Matrix, M2 : Matrix) : Matrix</u> <u>+ operator *(M1 : Matrix, M2 : Matrix) : Matrix</u> <u>+ operator *(M1 : Matrix, Scalar : Float) : Matrix</u> <u>+ operator ^(M1 : Matrix) : Matrix</u>
+ ToString() : String + Transpose() : Matrix + SelectColumn(Col: Int) : List + SelectRow(Row: Int) : List + Sum() : Float + Max() : Float + Clear()

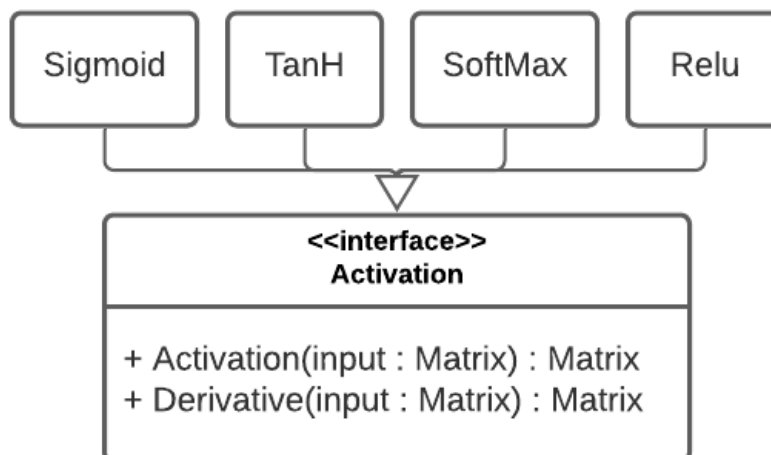
The Deque Class is used as part of the Experience Replay Algorithm.



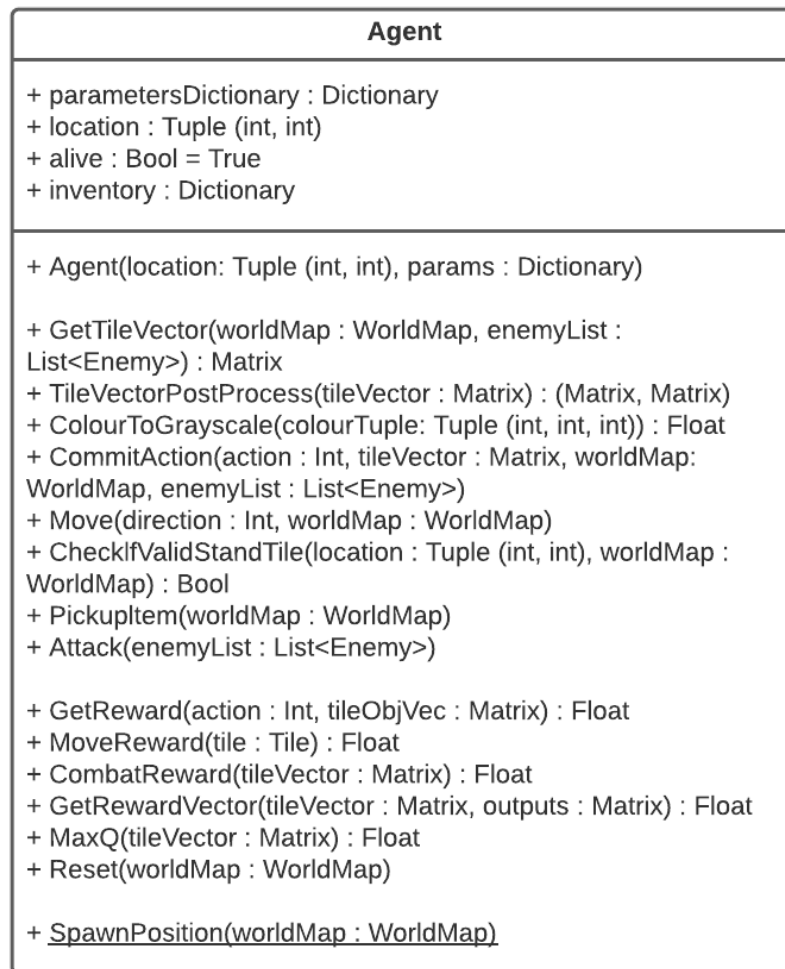
The Experience Class is stored within the Deque Object.



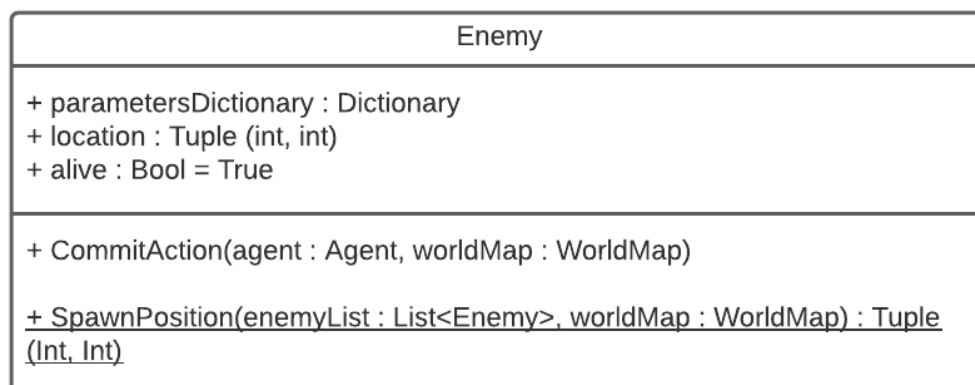
The Activation Class is an Abstract Base Class, in which the Neural Network Activations can inherit from, implementing their own definitions for Activation and Derivative.



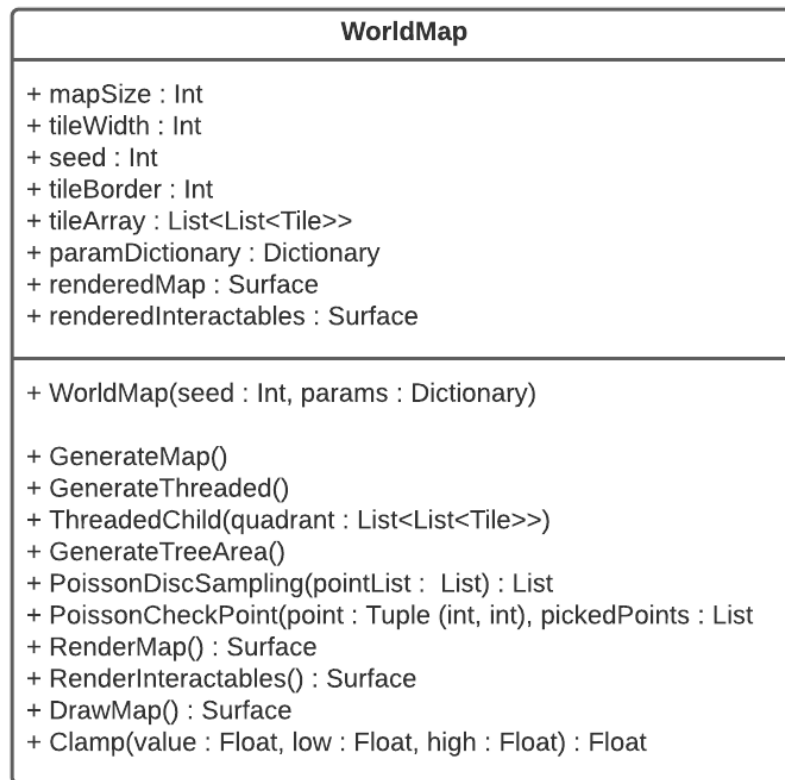
The Agent Class is used to store the Agents Location, along with implementing Action and Reward Methods.



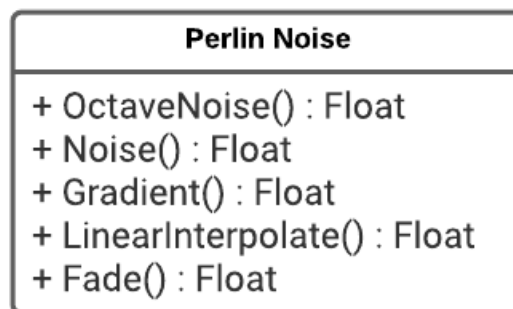
The Enemy Class inherits methods from Agent, Implementing its own CommitAction and SpawnPosition Methods.



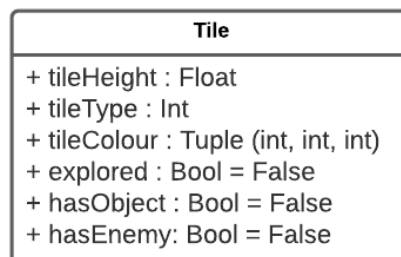
The WorldMap Class generates and stores all Terrain Data for the current Simulation.



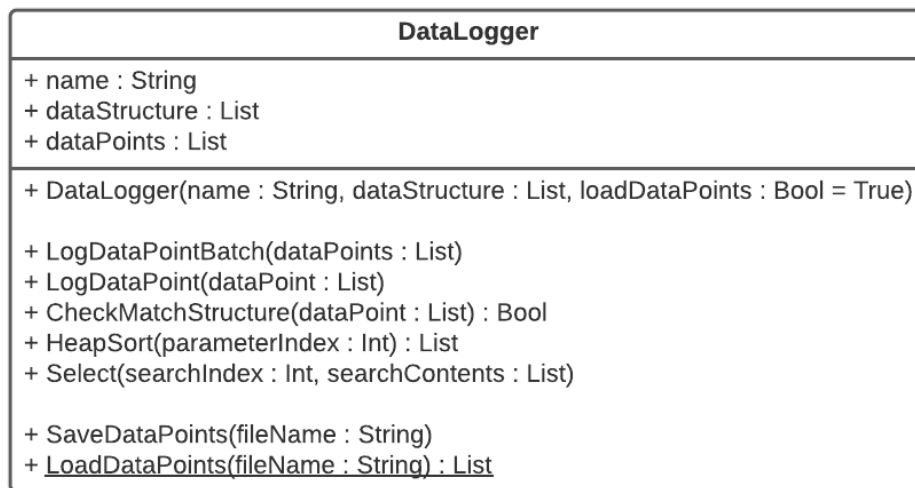
The Perlin Noise Class contains only methods and is used to Sample Gradient Noise based on a Seed.



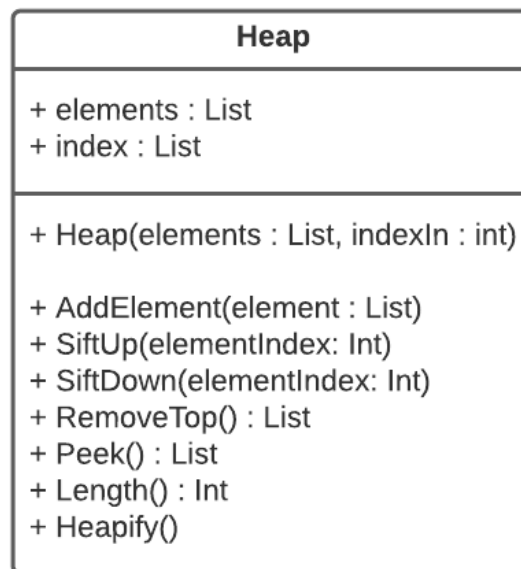
The Tile Class is used to Store and Manipulate Data per Tile.



The DataLogger Class is used to log Data Points across the Program.



The Heap Class is used as part of the Heap Sort implemented by the DataLogger.



2.4 Choice of Programming Language and Libraries

During my Analysis I outlined a list of possible Programming Languages and Associated Libraries. I chose Python and Pygame as part of my prototype. I found this combination to be very easy to use and iteratively develop my prototype.

2.5 Description of Algorithms

In this section, I will describe the algorithms I intend to use in my Technical Solution. I will also include generalised Pseudocode as part of my description.

2.5.1 Matrix Addition

This algorithm is a Mathematical Operation to add 2 Matrices together. To Add together 2 Matrices their Orders must be the same. To perform the Operation you must Sum each element in Matrix A with the corresponding element in Matrix B, placing the result of each Sum in the resultant Matrix.

```

1 | SUBROUTINE MatrixAddition(Matrix1, Matrix2)
2 |     TemporaryMatrix ← NEW Matrix(Matrix1.Order)
3 |     FOR Row ← 0 TO Matrix1.Order[0]
4 |         FOR Column ← 0 TO Matrix1.Order[1]
5 |             TemporaryMatrix[Row, Column] ← Matrix1[Row, Column] + Matrix2[Row, Column]
6 |         END FOR
7 |     END FOR
8 |     RETURN TemporaryMatrix
9 | ENDSUBROUTINE

```

2.5.2 Matrix Subtraction

This algorithm is a Mathematical Operation to subtract 2 Matrices. To Subtract 2 Matrices their Orders must be the same. To perform the Operation you must Sum each element in Matrix A with the negative of the corresponding element in Matrix B, placing the result of each Sum in the resultant Matrix.

```

1 | SUBROUTINE MatrixSubtraction(Matrix1, Matrix2)
2 |     TemporaryMatrix ← NEW Matrix(Matrix1.Order)
3 |     FOR Row ← 0 TO Matrix1.Order[0]
4 |         FOR Column ← 0 TO Matrix1.Order[1]
5 |             TemporaryMatrix[Row, Column] ← Matrix1[Row, Column] - Matrix2[Row, Column]
6 |         END FOR
7 |     END FOR
8 |     RETURN TemporaryMatrix
9 | ENDSUBROUTINE

```

2.5.3 Matrix Multiplication

This algorithm is a Mathematical Operation to find the product of 2 Matrices. To Multiply 2 Matrices the number of Columns in the Matrix A must be equal to the number of Rows in Matrix B. Where Matrix A has dimensions of $m \times n$ and Matrix B has dimensions of $j \times k$, the resultant Matrix will have dimensions of $n \times j$. To Multiply two Matrices, the algorithm performs the Dot Product between the Row in Matrix A and the corresponding Column in Matrix B. The Dot Product is the Sum of the Products of corresponding elements.

```

1 | SUBROUTINE MatrixMultiplication(Matrix1, Matrix2)
2 |     tempMatrix ← NEW Matrix((Matrix1.Order[0], Matrix2.Order[1]))
3 |     FOR i ← 0 TO Matrix1.Order[0]
4 |         FOR j ← 0 TO Matrix2.Order[1]
5 |             FOR l ← 0 TO Matrix1.Order[1]
6 |                 tempMatrix[i, j] ← tempMatrix[i, j] + Matrix1[i, l] * Matrix2[l, j]
7 |             END FOR
8 |         END FOR
9 |     END FOR
10 |    RETURN tempMatrix
11 | ENDSUBROUTINE

```

2.5.4 Matrix Scalar Multiplication

This algorithm is a Mathematical Operation to find the product between a Matrix and a Scalar. The result can be found by Multiplying each element of the Matrix by the Scalar Value to form the Resultant Matrix.

```

1 | SUBROUTINE MatrixScalarMultiplication(Scalar, Matrix)
2 |     TemporaryMatrix ← NEW Matrix(Matrix.Order)
3 |     FOR Row ← 0 TO Matrix.Order[0]
4 |         FOR Column ← 0 TO Matrix.Order[1]
5 |             TemporaryMatrix[Row, Column] ← Scalar * Matrix[Row, Column]
6 |         END FOR
7 |     END FOR
8 |     RETURN TemporaryMatrix
9 | ENDSUBROUTINE

```

2.5.5 Matrix Hadamard Product

This algorithm is a Mathematical Operation to another way to find the product between 2 Matrices. Instead of applying the Dot Product between Rows and Columns, you find the product between each element in Matrix A with the corresponding element in Matrix B, placing the result in the resultant Matrix.

```

1 | SUBROUTINE MatrixHadamardProduct(Matrix1, Matrix2)
2 |     TemporaryMatrix ← NEW Matrix(Matrix1.Order)
3 |     FOR Row ← 0 TO Matrix1.Order[0]
4 |         FOR Column ← 0 TO Matrix1.Order[1]
5 |             TemporaryMatrix[Row, Column] ← Matrix1[Row, Column] * Matrix2[Row, Column]
6 |         END FOR
7 |     END FOR
8 |     RETURN TemporaryMatrix
9 | ENDSUBROUTINE

```

2.5.6 Matrix Power

This algorithm is a Mathematical Operation to find the power of a Matrix. The given Matrix needs to have square dimensions. The result can be found by multiplying the given Matrix by itself n ammount of times where n is the given power.

```

1 | SUBROUTINE MatrixHadamardProduct(Matrix, Power)
2 |     TemporaryMatrix ← CLONE Matrix
3 |     FOR Row ← 0 TO Power - 1
4 |         TemporaryMatrix ← TemporaryMatrix * Matrix
5 |     END FOR
6 |     RETURN TemporaryMatrix
7 | ENDSUBROUTINE

```

2.5.7 Matrix Transpose

This algorithm is a Mathematical Operation used to Flip a Matrix across its Diagonal. The Transpose of any Matrix can be found by converting each Row of the Matrix into a Column. An $m \times n$ Matrix will turn into an $n \times m$ Matrix.

```

1 | SUBROUTINE MatrixTranspose(Matrix)
2 |     TemporaryMatrix ← NEW Matrix(Matrix.Order)
3 |     FOR Row ← 0 TO Matrix.Order[0]
4 |         FOR Column ← 0 TO Matrix.Order[1]
5 |             TemporaryMatrix[Row, Column] ← Matrix[Column, Row]

```



```

6      END FOR
7      END FOR
8      RETURN temporaryMatrix
9  ENDSUBROUTINE

```

2.5.8 Activation Function SoftMax

This algorithm is a logistic function that creates a probability distribution from a set of points. This probability distribution sums to 1. It applies the standard Exponential Function to each element, then normalises this value by dividing by the sum of all these Exponentials.

```

1  SUBROUTINE Softmax(Input)
2      OutVector ← NEW Matrix(Input.Order)
3      ExpSum ← 0
4      FOR Row ← 0 TO Input.Order[0]
5          ExpSum ← ExpSum + Math.exp(Input[Row, 0])
6      END FOR
7      FOR Row ← 0 TO Input.Order[0]
8          OutVector[Row] ← Input[Row, 0] / ExpSum
9      END FOR
10     RETURN OutVector
11 ENDSUBROUTINE

```

2.5.9 Neural Network Forward Propagation

This algorithm is used to obtain the outputs of a Neural Network. It uses Matrix Multiplication to propagate the inputs of the network from Layer to Layer, eventually reaching the Output Layer. My Multiplying the Weight Matrix and the outputs of the previous Layer, and then adding the Bias. We can obtain the output of the layer.

```

1  SUBROUTINE Forward Propagation(PrevLayer, Activations, FinalLayer)
2      WeightValueProduct ← This.WeightMatrix * PrevLayer.OutputVector
3      This.SVector ← WeightValueProduct + This.BiasVector
4      IF NOT FinalLayer
5          This.OutputLayer ← Activations[0].Activation(SVector)
6      ELSE
7          This.OutputLayer ← Activations[1].Activation(SVector)
8      END IF
9  ENDSUBROUTINE

```

2.5.10 Half Square Difference

This algorithm is the Cost Function of the Neural Network used in tandem with Bellman Equation. It takes the network output away from the expected value, squares and then halves it, per output node.

```

1  SUBROUTINE HalfSquareDiff(NetworkOutput, Expected)
2      RETURN 0.5 * Math.pow((Expected - NetworkOutput, 2)
3  ENDSUBROUTINE

```

2.5.11 Neural Network Bellman Equation

This algorithm calculates the expected value of the Neural Network. This is calculated using a variation of the Bellman Equation. The Bellman Equation is necessary for Mathematically Optimising in this case. It determines the Value of a decision at a certain point in time, in terms of the Payoff from the Initial Action and the Value of the Potential Payoff after taking that Initial Action.

1

2.5.12 Neural Network Backwards Propagation

This algorithm is used within a Neural Network to adjust its Weights and Biases, allowing it to more accurately predict the best outcome. In Reinforcement Learning, the Network is trained using an estimate for what is the best action given a situation. Using this estimate, we can train the Network to predict this outcome by converging the series of Weights and Biases towards a local minimum. This is done by calculating partial derivatives for every weight and bias value with respect to the cost function. This derivative is then subtracted from the existing weight or bias, eventually converging on the best possible value.

```

1  SUBROUTINE BackPropagation(PreviousLayer, LearningRate, Activation)
2      WeightTranspose ← PreviousLayer.WeightMatrix.Transpose()
3      DeltaWeightProduct ← WeightTranspose * PreviousLayer.ErrorSignal
4      This.ErrorSignal ← DeltaWeightProduct * Activation.Derivative(This.PreActivations)
5
6      WeightDerivatives ← This.ErrorSignal * This.Activations.Transpose()
7      BiasDerivatives ← This.ErrorSignal
8
9      This.WeightUpdates ← This.WeightUpdates + (WeightDerivatives * LearningRate)
10     This.BiasUpdates ← This.BiasUpdates + (BiasDerivatives * LearningRate)
11 ENDSUBROUTINE

```

2.5.13 Experience Replay

This algorithm samples a Double Ended Queue of (State, Action, Reward, State') Tuples and performs the Back Propagation Algorithm on the data. This process is designed to immitate the recall of previous experiences stored in the agents figurative Memory.

```

1  SUBROUTINE ExperienceReplay(SampleSize, Agent)
2      Samples ← NEW List()
3      FOR i ← 0 TO SampleSize
4          Samples.Add(Buffer.RandomSample())
5      END FOR
6
7      FOR Sample IN Samples
8          PostProcessedSurround ← Agent.TileVectorPostProcess(sample.state)
9
10         NetInput ← PostProcessedSurround[1]
11
12         This.MainNetwork.ForwardPropagation(NetInput, This.Activation)
13
14         Output ← This.MainNetwork.Layers[-1].Activations
15
16         ExpectedValues ← This.ExpectedValue(Output, Sample, Agent)

```

```

17      Cost ← This.HalfSquareDiff(Output, ExpectedValues)
18
19      Preactivations ← This.MainNetwork.Layers[-1].Preactivations
20      PreactivationsDerivative ← This.Activation.Derivative(Preactivations)
21      This.MainNetwork.Layers.ErrSignal ← Cost * PreactivationsDerivative
22
23      This.MainNetwork.BackPropagation(This.Activation)
24  END FOR
25  ENDSUBROUTINE
26

```

2.5.14 Agent Get Tile Vector

This algorithm takes the current World Data of the simulation, and produces a Vector of Tile Data surrounding the Agent. This can be done using a nested For Loop rather simply.

```

1  SUBROUTINE GetTileVector(WorldMap)
2      Offset ← LoadFromParameters("DQLOffset")
3      SideLength ← 2 * Offset + 1
4      TileVector ← NEW Matrix((Math.pow(sideLength, 2), 1))
5      Num ← 0
6      FOR i ← Agent.Pos[1] - Offset TO Agent.Pos[1] + Offset + 1
7          FOR j ← Agent.Pos[0] - Offset TO Agent.Pos[1] + Offset + 1
8              TileVector[Num, 0] ← WorldMap[j, i]
9              Num ← Num + 1
10         END FOR
11     END FOR
12     RETURN TileVector
13 ENDSUBROUTINE

```

2.5.15 Agent Convert to Grayscale

This algorithm converts a given RGB Colour Value to the corresponding Gray Scale Value. The Red, Green and Blue elements of the colour value are multiplied by the specific values 0.299, 0.587 and 0.114. You then sum the results, and divide by 255.

```

1  SUBROUTINE RGBToGrayscale(RGBVal)
2      GrayscaleValue ← 0
3      GrayscaleValue ← GrayscaleValue + (0.299 * RGBVal[0])
4      GrayscaleValue ← GrayscaleValue + (0.587 * RGBVal[1])
5      GrayscaleValue ← GrayscaleValue + (0.114 * RGBVal[2])
6      RETURN GrayscaleValue / 255
7  ENDSUBROUTINE

```

2.5.16 Agent Post Process Tile Vector

This algorithm will convert the Tile Vector into a Vector of Grayscale values, which can be used as the input for the Neural Network.

```

1  SUBROUTINE GetTileVector(TileVector)
2      ProcessedVector ← NEW Matrix(TileVector.Order)
3      FOR Row ← 0 TO TileVector.Order[0]
4          ProcessedVector[Row, 0] ← RGBToGrayscale(TileVector[Row, 0].RGBValue)

```

```

5 |     END FOR
6 |     RETURN ProcessedVector
7 | ENDSUBROUTINE

```

2.5.17 Agent Spawn Position

This algorithm will create a list of spawnable tiles for which the Agent could spawn on, and then randomly select a specific tile as its spawn position.

```

1 | SUBROUTINE AgentSpawnPosition(WorldMap)
2 |     SpawnList ← NEW List()
3 |     MapSize ← LoadFromParameters("MapSize")
4 |     FOR y ← 0 TO MapSize
5 |         FOR x ← 0 TO MapSize
6 |             IF WorldMap[x, y].TileType == 2
7 |                 SpawnList.Add([x, y])
8 |             END IF
9 |         END FOR
10 |     END FOR
11 |     SpawnList.Shuffle()
12 |     RETURN SpawnList[0]
13 | ENDSUBROUTINE

```

2.5.18 Enemy Spawn Position

This algorithm will create a list of spawnable tiles for which Enemies can spawn on, then select tiles randomly, if they don't already contain an enemy or the agent it will create an Enemy Object with that position. It will do this n amount of times where n is the limit to how many enemies can spawn.

```

1 | SUBROUTINE EnemySpawnPosition(WorldMap, EnemyList)
2 |     SpawnList ← NEW List()
3 |     EnemyLocationList ← NEW List()
4 |     MapSize ← LoadFromParameters("MapSize")
5 |     FOR y ← 0 TO MapSize
6 |         FOR x ← 0 TO MapSize
7 |             IF WorldMap[x, y].TileType == 2
8 |                 SpawnList.Add([x, y])
9 |             END IF
10 |         END FOR
11 |     END FOR
12 |     SpawnList.Shuffle()
13 |     IF SpawnList[0] IN EnemyLocationList
14 |         RETURN NONE
15 |     ELSE
16 |         RETURN SpawnList[0]
17 |     END IF
18 |     RETURN SpawnList[0]
19 | ENDSUBROUTINE

```

2.5.19 Enemy Move

The algorithm I have designed for the Enemy Pathfinding is rather simple, and wont take up much runtime in my solution. First it calculates the distance between itself and the Agent in both Axis. The Enemy will then converge upon the Agents position by moving in the direction with the greatest distance, effectively finding the nearest diagonal and following it.

```

1  SUBROUTINE EnemyMove(Agent, WorldMap)
2      XDifference ← Agent.Pos[0] - This.Pos[0]
3      YDifference ← Agent.Pos[1] - This.Pos[0]
4
5      IF XDifference == 0 AND YDifference == 0
6          Agent.Alive = False
7          RETURN
8      END IF
9
10     IF abs(XDifference) > abs(YDifference)
11         IF XDifference > 0
12             This.Pos[0] ← This.Pos[0] + 1
13         ELSE
14             This.Pos[0] ← This.Pos[0] - 1
15         END IF
16     ELSE IF abs(XDifference) < abs(YDifference)
17         IF YDifference > 0
18             This.Pos[1] ← This.Pos[1] + 1
19         ELSE
20             This.Pos[1] ← This.Pos[1] - 1
21         END IF
22     END IF
23 ENDSUBROUTINE

```

2.5.20 Poisson Disc Sampling

Poisson Disc Sampling is used to sample a set of points in N Dimensional Space. It takes two parameters, r and k , where r is the minimum distance a specified point must be from every other point, and k is the limit of samples to choose before rejection. It starts by creating an N Dimensional Grid which accelerates spacial searches. An initial sample is then chosen and inserted into the grid. It then chooses a random point, and determines if it is greater than r range from every other point in the grid. This can easily be acomplished using the previously defined Grid. If after k attempts, no point is found then the search is concluded.

```

1  SUBROUTINE PoissonDiscSampling(PointList)
2      KVal ← LoadFromParameters("PoissonKVal")
3      MapSize ← LoadFromParameters("MapSize")
4      PickedPoints ← NEW Grid(MapSize, MapSize)
5      SampleNum ← LoadFromParameters("MapSize")
6      WHILE SampleNum <= KVal
7          Sample ← PointList[RandomInt(0, PointList.Length - 1)]
8          Result ← CheckPointDistance(Sample, PickedPoints)
9          IF Result == True
10             PickedPoints[Sample[0], Sample[1]] ← True
11             SampleNum ← 0
12             CONTINUE
13         ELSE
14             SampleNum ← SampleNum + 1
15             CONTINUE
16         END IF

```

```

17     END WHILE
18     RETURN PickedPoints
19 ENDSUBROUTINE

```

2.5.21 Perlin Noise

Perlin Noise is a method of generating a procedural texture depending upon input parameters. It defines an n-dimensional grid of Vectors, each grid intersection contains a fixed, random unit vector. To sample Perlin Noise, the grid cell which the point lies in must be found. The Vectors between the sampled point, and the corners of the cell. We then take the Dot Product between these new Vectors, and the Vectors applied to the intersections. In 2d Space this leaves us with 4 Values. We then use an Interpolation function to Interpolate between the 4 Values.

```

1  PermTable ← [1 → 255].Shuffle() * 2
2
3  SUBROUTINE PerlinNoise(X, Y)
4      XFloor ← Math.floor(X)
5      YFloor ← Math.floor(Y)
6
7      G1 ← PermTable[PermTable[XFloor] + YFloor]
8      G2 ← PermTable[PermTable[XFloor + 1] + YFloor]
9      G3 ← PermTable[PermTable[XFloor] + YFloor + 1]
10     G4 ← PermTable[PermTable[XFloor + 1] + YFloor + 1]
11
12     XExact ← X - XFloor
13     YExact ← Y - YFloor
14
15     D1 ← Grad(G1, XFloor, YFloor)
16     D2 ← Grad(G2, XFloor - 1, YFloor)
17     D3 ← Grad(G3, XFloor, YFloor - 1)
18     D4 ← Grad(G4, XFloor - 1, YFloor - 1)
19
20     U ← Fade(XFloor)
21     V ← Fade(YFloor)
22
23     XInterpolated ← Lerp(U, D1, D2)
24     YInterpolated ← Lerp(U, D3, D4)
25
26     RETURN Lerp(V, XInterpolated, YInterpolated)
27 ENDSUBROUTINE
28
29 SUBROUTINE Grad(Hash, X, Y)
30     Temp ← Hash BITWISEAND 3
31     IF Temp == 0
32         RETURN X + Y
33     ELSE IF Temp == 1
34         RETURN -X + Y
35     ELSE IF Temp == 2
36         RETURN X - Y
37     ELSE IF Temp == 3
38         RETURN -X - Y
39     ELSE
40         RETURN 0
41     END IF
42 ENDSUBROUTINE
43

```

```

44 | SUBROUTINE Lerp(Amount, Left, Right)
45 |     RETURN ((1 - Amount) * Left + Amount * Right)
46 | ENDSUBROUTINE
47 |
48 | SUBROUTINE Fade(T)
49 |     RETURN T * T * T * (T * (T * 6 - 15) + 10)
50 | ENDSUBROUTINE

```

2.5.22 Octave Perlin Noise

Octave Perlin Noise takes the existing Perlin Noise algorithm, but adds rescaled clones of itself into itself, to create what is known as Fractal Noise. Creating this Fractal Noise is common practice because it reduces the sharp edges encountered with just the regular Perlin Noise Algorithm.

```

1 | SUBROUTINE OctaveNoise(X, Y, Octaves, Persistence)
2 |     Total ← 0
3 |     Frequency ← 1
4 |     Amplitude ← 1
5 |     MaxValue ← 0
6 |
7 |     FOR i ← 0 TO Octaves
8 |         Total ← Total + (PerlinNoise(X * Frequency, Y * Frequency) * Amplitude
9 |
10 |         MaxValue ← MaxValue + Amplitude
11 |
12 |         Amplitude ← Amplitude * Persistence
13 |         Frequency ← Frequency * 2
14 |     END FOR
15 |
16 |     RETURN Total / MaxValue
17 | ENDSUBROUTINE

```

2.5.23 Heap Heapify

The Heapify algorithm converts a Binary Tree of values into a valid Heap. The Heap Property is defined in Description of Data Structures below. This algorithm works by repeatedly performing Sift Down Operations for $\lfloor (N-1)/2 \rfloor$ times. Where N is the Number of elements in the Tree. A Sift Down Operation will swap elements which don't conform to the Heap Property. This operation relies on the fact that Children of an Index are located at $2i + 1$ and $2i + 2$.

```

1 | SUBROUTINE Heapify()
2 |     FOR i ←  $\lfloor (\text{HeapList.Length}-1)/2 \rfloor$  TO 0 STEP -1
3 |         SiftDown(i)
4 |     END FOR
5 | ENDSUBROUTINE
6 |
7 | SUBROUTINE SiftDown(RootIndex)
8 |     IsHeap ← FALSE
9 |     End ← HeapList.Length - 1
10 |
11 |     WHILE (2 * RootIndex) + 1 <= End
12 |         ChildIndex = (RootIndex * 2) + 1
13 |         IF ChildIndex <= End AND HeapList[ChildIndex] < HeapList[ChildIndex + 1]
14 |             ChildIndex ← ChildIndex + 1
15 |         END IF

```

```

16     IF HeapList[RootIndex] < HeapList[ChildIndex]
17         TempSwap ← HeapList[ChildIndex]
18         HeapList[ChildIndex] ← HeapList[RootIndex]
19         HeapList[RootIndex] ← TempSwap
20     ELSE
21         BREAK
22     END IF
23 ENDSUBROUTINE

```

2.5.24 Heap Extraction

This algorithm extracts the Root Element from a valid Heap. It does this by swapping the Root Element and Final Element, and then popping the new Final Element (Originally the Root) from the list.

```

1 SUBROUTINE RemoveTop()
2     TempSwap ← HeapList[-1]
3     HeapList[-1] ← HeapList[0]
4     HeapList[0] ← TempSwap
5     ReturnItem ← HeapList.Pop()
6
7     Heapify()
8
9     RETURN ReturnItem
10 ENDSUBROUTINE

```

2.5.25 Heap Sort

The Heap Sort algorithm relies on the prior two algorithms to fully order a list in Worst and Best case $O(n \log(n))$ Time Complexity. It is also $O(1)$ Space Complexity due to it being an In-Place Sorting algorithm. The sort will iteratively shrink the unsorted region by performing the following steps: Apply Heapify to the Unsorted Region, Extract the Root Element from the Heap, Insert the Extracted Element at the end of the Unsorted Region. This allows it to be In-Place because it never requires extra space.

```

1 SUBROUTINE HeapSort()
2     SortedList ← NEW List()
3     Heap ← NEW Heap(DataPoints)
4
5     WHILE Heap.Size() - 1 >= 0
6         SortedList.Append(Heap.RemoveTop())
7     END FOR
8
9     RETURN SortedList
10 ENDSUBROUTINE

```

2.6 Description of Data Structures

1. Matrices

As part of developing a Neural Network, I will extensively use Matrices, as they are an integral part of the algorithms used for Machine Learning. After creating a prototype

Matrix class as part of my prototype, I will represent it in the same format. A Matrix can be represented simply using a 2D Array, but they can have Mathematical Operations performed between them. Explanations and the formulae can be found in the Modelling of the Problem Analysis Section.

To avoid repeating code in some places, Matrices will have multiple Constructors. The main Constructors are in the form of an (Int, Int) Tuple, or an pre-existing 2D Array. Other less used examples could be an Integer for creating a Vector of that length.

Operator Overloading will be useful when implementing a Matrix Class, as it allows classes to have implementations for operators such as Multiplication, Addition, Subtraction etc. This avoids the need to rely on Static Methods for Operator Implementations and makes code much more readable overall.

As part of a Neural Network Matrices are used heavily in the calculations. So it will be important to optimise the implemented algorithms to make sure their Algorithmic Time Complexity is minimised.

2. Double Ended Queue

A Double Ended Queue (Commonly referred to as a **Deque**) is an Abstract Data Type, which is a generalisation of a Queue. Elements can be added to the Front/Head or Back/Tail. Deques are commonly implemented using an Array, and two pointers, one for Front and Back.

3. Tile

A Tile is used to store specific location Data as part of the World Map. It can be initialised without values, and is then populated with the relevant information. Methods are attached to this Class to Add/Remove Items and Enemies as needed. Allowing for the Agent when getting Tile data to get relevant and accurate information.

4. Experience

An Experience is used to store data for Experience Replay. It is an Empty Class with no Methods. This includes the State, Action, NewState and Reward, all at the time of assignment. This is used in conjunction with the Experience Replay Algorithm, described above.

5. Heap

A Heap is specialised Binary Tree which satisfies the **Heap Property**: such that for all nodes with Parents, the Parent has a greater value than the Child. A Heap is used as part of a Heap Sort, an $O(n \log(n))$ Sorting Algorithm. The highest priority element is always stored at the Root, with the tree of the structure being considered "Partially Ordered". Heaps can be stored in an Array, with the Root element at Index 0. Children of an Index are located at $2i + 1$ and $2i + 2$. The Parent of an Index is located at $\lfloor (i-1)/2 \rfloor$.

2.7 File Structure

1. User Defined Parameters

As part of my Technical Solution, the User will be able to modify the parameters which dynamically modifies the Simulation and the Structure of the Double Neural Network.

The file is stored in a Json format (Java Script Object Notation). This allows the File to be Human Readable, and easily editable. Each parameter will also have a defined Range alongside it. The program will throw an error if the parameter is outside the specified range. Below is a table of the Parameters used in the Technical Solution, alongside their respective Ranges.

Name in Json	Data Type	Range	Description
EnterValues	Int	0 - 1	The program will ask you to enter values if this is 1
GenerateThreaded	Int	0 - 1	The program will generate the Terrain using Multiple Threads
EnableEnemies	Int	0 - 1	Toggled Enable Enemies Option.
SaveWeights	Int	0 - 1	Toggled Save Network Weights Option.
StepDelay	Float	0 - ∞	The time delay each step.
Debug	Int	0 - 1	Toggled Debug Option.
DebugScale	Int	1 - 4	The scale of the Debug side extension.
WorldSize	Int	16 - 1024	The size the of the World in Tiles. Must be a Multiple of 2.
TileWidth	Int	1 - 8	The Width and Height of each Tile.
TileBorder	Int	0 - 3	The Pixel Border surrounding Tiles.
OctavesTerrain	Int	1 - 20	The Perlin Noise Octave Value for World Generation.
PersistenceTerrain	Float	0 - 1	The Perlin Noise Persistence Value for World Generation.
WorldScale	Float	0.1 - 10	The Perlin Noise Scale Value for World Generation.
OctavesTrees	Int	1 - 20	The Perlin Noise Octave Value for Trees
PersistenceTrees	Float	0 - 1	The Perlin Noise Persistence Value for generating the Trees.
PoissonKVal	Int	0 - ∞	The K Value for Poisson Disc Sampling.
TreeSeedOffset	Int	0 - ∞	The Seed offset for generating the Trees.
TreeHeight	Float	0 - 1	The difference between Min Tree spawning height and Max Tree spawning height.
InteractableTileBorder	Int	0 - 3	The Pixel Border surrounding Interactables.
TreeBeachOffset	Float	0 - 1	The height difference from Beaches which Trees will Spawn.
Grayscale	Int	0 - 1	Toggled Grayscale Terrain Option.
Water	Float	0 - 1	The cutoff values for Water.
Coast	Float	0 - 1	The cutoff values for Coast.
Grass	Float	0 - 1	The cutoff values for Grass.
Mountain	Float	0 - 1	The cutoff values for Mountains.
TreeType	String	0 - 1	The internally used Inventory name for collected Trees.
StartEnemyCount	Int	0 - ∞	The maximum count of Enemies to Spawn upon the creation of a new Map.
ColourWater	[Int, Int, Int]	0 - 255	The display Colour of Water.
ColourCoast	[Int, Int, Int]	0 - 255	The display Colour of Coast.
ColourGrass	[Int, Int, Int]	0 - 255	The display Colour of Grass.
ColourMountain	[Int, Int, Int]	0 - 255	The display Colour of Mountains.
ColourTree	[Int, Int, Int]	0 - 255	The display Colour of Trees.
ColourPlayer	[Int, Int, Int]	0 - 255	The display Colour of the Agent.
ColourEnemy	[Int, Int, Int]	0 - 255	The display Colour of Enemies.
MoveReward	Float	-1 - 1	The Reward Gained when the Agent Moves.
CollectItemReward	Float	-1 - 1	The Reward Gained when the Agent collects an Item.
DeathReward	Float	-1 - 1	The Reward Gained when the Agent Dies through any means.
ExploreReward	Float	-1 - 1	The Reward Gained when the Agent moves into a Tile which hasnt been Visited yet.

AttackReward	Float	-1 - 1	The Reward Gained when the Agent successfully Attacks an Enemy.
AttackFailedReward	Float	-1 - 1	The Reward Gained when the Null Action is chosen.
NoopReward	Float	-1 - 1	The Reward Gained when the Null Action is chosen.
TargetReplaceRate	Int	5 - 300	Replace Rate for Target Neural Network.
EREnabled	Int	0 - 1	Whether Experience Replay is Enabled or Disabled.
ERBuffer	Int	1k - 10k	The size of the Experience Replay Buffer.
ERSampleRate	Int	1 - 100	The amount of steps between each Experience Replay sample.
ERSampleSize	Int	10 - 1000	The amount of samples taken from the Experience Replay Buffer.
DeepQLearningLayers	[Int, ..., Int]	0 - 256	List of Integers defining the size of each Layer in the Neural Network.
DQLEpoch	Int	10 - 1000	The amount of steps per Weight and Bias Update, along with Network Saving and Debug Output
DQLearningMaxSteps	Int	1000 - ∞	Maximum steps the Simulation will run for.
DQLOffset	Int	1 - 10	The square radius around the agent which is sampled for the Input vector, must be the root of the Input Layers size.
DQLEpsilon	Float	0 - 1	The initial Probability that the Agent will favour a Random Action over the predicted Action
DQLEpsilonRegression	Float	0 - 1	The rate at which Epsilon will decrease, Epsilon is multiplied every step by this number
DQLLearningRate	Float	0 - 1	The Learning Rate of the Neural Network. Higher values will cause more drastic changes during Back Propagation.
DQLGamma	Float	0 - 1	The Discount for future gained Reward

2. .dqn Files

DQN Files are used to store all Data relating to the Dual Neural Network. It is a Binary File. It contains all Layer Data, along with Experience Replay Data, the activations being used, and other important data.

3. .data Files

Data Files are used to store all data points created by the Data Loggers. They are Binary Files and are individually created per Data Logger.

3 Testing

3.1 Testing Table

3.1.1 Targetted Testing Areas

As part of testing my NEA, I identified the key areas of my project which needed testing. My testing targets these areas from different angles to ensure they work correctly. These areas are:

1. User Input and Program Output
 - (a) Parameter Loading
 - (b) Neural Network Loading
 - (c) Graphical Output
 - (d) Console Output
2. Matrix Implementation
 - (a) Constructor Cases
 - (b) Matrix Operations
 - (c) Thrown Exceptions
3. Deep Q Learning Algorithm
 - (a) Forward Propagation
 - (b) Loss Function
 - (c) Back Propagation
 - (d) Double Ended Queue Data Type
4. Data Logger
 - (a) Data Structure Matching
 - (b) Heap Data Structure
 - (c) Heap Sort Implementation
5. Simulation
 - (a) Generation of 2d Terrain
 - (b) Continuity of Generation
 - (c) ML Agent
 - (d) Reward Methods

Below is included an NEA Testing video used for some parts of Testing Evidence

<https://thisisalink.com/youtotallybelieveme/>

3.1.2 User Input and Program Output Tests

Test No.	Test Name	Input Data / Description	Expected Output	Pass / Fail	Testing Evidence
1	Loading Parameters File	Input "Default.json" file which contains the loadable values	Loads parameters into the Parameters Dictionary variable	Pass	1.1
2	Parameters within range	Input Loaded Parameters Dictionary	Prints to console "Parameters within Specified Ranges"	Pass	1.2
3	Below Range Parameter	Input "Default.json" file with a below range parameters	Raises an exception detailing the Parameter, Value of Parameters, and the given Range Required	Pass	1.3
4	Above Range Parameter	Input "Default.json" file with an above range parameters	Raises an exception detailing the Parameter, Value of Parameters, and the given Range Required	Pass	1.4
5	Network Saved Data Loading	When Prompted to load network data type "Y", and type the file name of network data to load	Network Data is loaded successfully, training position stored	Pass	1.5
6	Window Opening	Run Program, enter setup info as normal	Window opens and is of the correct size/resolution	Pass	1.6
7	Window Displays correct debug information	Run Program, enter setup info as normal, with "Debug" = 1 in parameters file	Debug Layer output info displayed on Right side of Window	Pass	1.7
8	Agent is displayed	Run Program, enter setup info as normal	Orange square displayed on screen	Pass	1.8
9	Enemies are displayed	Run Program, enter setup info as normal, with "StartEnemyCount" >= 1	Red Square/s are displayed on Screen	Pass	1.9
10	Console Messages Output	Run Program, enter setup info as normal	Console Messages Outputted per 100 Steps	Pass	1.10

3.1.3 Matrix Implementation Tests

Test No.	Test Name	Input Data / Description	Expected Output	Pass / Fail	Testing Evidence
1	Create Matrix with Tuple	A Tuple for the order of the Matrix	Matrix is created with an order the same as the Tuple	Pass	2.1

2	Create Matrix with 2d List	A 2d List, where the parent list holds a list for every row, each "row list" is of the same length	Matrix is created with the same values as the 2d List	Pass	2.2
3	Create Vector with List	A 1d List of any Values	Vector is created with the same values as the List	Pass	2.3
4	Print Matrix to Console	A valid Matrix of any size	Matrix Prints to the console with the correct formatting	Pass	2.4
5	Create Randomised Matrix	A Tuple for the order of the Matrix, and the the keyargument random=True	Matrix is created with randomised values between -0.5 and 0.5	Pass	2.5
6	Create Identity Matrix	A Tuple for the order of the Matrix, and the the keyargument identity=True	Matrix is created with all 0's and 1's down the diagonal	Pass	2.6
7	Matrix Addition Calculation	Two Matrices of the same order	Matrix Addition is performed to create a new Matrix with the added values	Pass	2.7
8	Matrix Subtraction Calculation	Two Matrices of the same order	Matrix Subtraction is performed to create a new Matrix with the subtracted values	Pass	2.8
9	Matrix Multiplication Calculation	Two Matrices where Width of M1 is equal to the height of M2	Matrix Multiplication is performed to create a new Matrix with the multiplied values	Pass	2.9
10	Matrix Scalar Multiplication Calculation	A <i>float/int</i> as the scalar and any size Matrix	Matrix Scalar Multiplication is performed to create a new Matrix with the multiplied values	Pass	2.10
11	Vector Hadamard Product Calculation	Two Vectors with the same Order	Vector Hadamard Product is performed to create a new Vector with the multiplied values	Pass	2.11
12	Matrix Power Calculation	A Square Matrix with values stored in it	Matrix to the Power of is performed to create a new Matrix with the correct values	Pass	2.12
13	Matrix Transpose Calculation	A Matrix with values stored in it	New Matrix is created with values flipped across the diagonal	Pass	2.13
14	Matrix Select Column	A Matrix with values stored in it	Selects the indexed Column from the Matrix, returning as a list	Pass	2.14
15	Matrix Select Row	A Matrix with values stored in it	Selects the indexed Row from the Matrix, returning as a list	Pass	2.15
16	Vector Max in Vector	A Vector	Returns Largest value in Vector	Pass	2.16
17	Matrix Clear	A Matrix with values stored in it	Clears Matrix of any values	Pass	2.17
18	Combine Vectors	List of Vectors of the same Order	Combines the list of Vectors into a Matrix	Pass	2.18
19	Matrix Sum	-	Sums all values in the Matrix returning a <i>float/int</i>	Pass	2.19
20	Randomised Matrix Constructor Tests	Generator Constructor Parameters randomly for 10000 Tests	All Tests Should produce a valid Matrix	Pass	2.16

21	Randomised Constructor Exception Tests	Generate Random Data to cause Exceptions within the Constructor for 10000 Tests	All Tests should trigger the Targetted Exception for that test	Pass	2.17
22	Randomised Operator Tests	Generator Random Data to test the Operator Methods for 10000 Tests	All Tests should produce the correct result	Pass	2.18
23	Randomised Operator Exception Tests	Generate Random Data to cause Exceptions within the Operators for 10000 Tests	All Tests should trigger the Targetted Exception for that test	Pass	2.19

3.1.4 Deep Reinforcement Learning Algorithm Tests

Test No.	Test Name	Input Data / Description	Expected Output	Pass / Fail	Testing Evidence
1	Networks are Created	Run Program, enter setup info, denying the loading of weights	A Dual Neural Network is created after Program Start	Pass	3.1
2	Networks conforms to Parameters	Run Program, enter setup info, denying the loading of weights	The created Dual Neural Network conforms to the specified structure in the parameter "DeepQLearningLayers"	Pass	3.2
3	Forward Propagation Test	Where L is the Current Layer, Forward Propagation requires: $OutputVector^{L-1}$, $WeightMatrix^{L-1}$, $BiasVector^L$	The output of the Layer	-	-
4	Forward Propagation Multi Layer Test	Same as Entry Above	-	-	-
5	Loss Function Bellman Equation	-	-	-	-
6	Back Propagation Test	-	-	-	-
7	Back Propagation Multi Layer Test	-	-	-	-
8	Deque Push Front	A value to push to the Deque	Item is pushed to front of Deque	Pass	3.8
9	Deque First/Last	Call the .First() or .Last() Method for a Deque Object	Returns item at Front/Last index of Deque	Pass	3.9
10	Deque Sample N Ammount of Items	Call the .Sample(int N) Method, with a parameter of N items, for a Deque Object	Returns N number of random samples from Deque	Pass	3.10
11	Experience Replay Sampling	-	Back Propagation is performed on the sampled Deque Items	-	-
12	Activation Outputs Unit Test	Input Value Vector to the Activation Function	Returns a Vector of values, where the Activation has been applied to them	-	-
13	Activation Derivatives Output Unit Test	Input Value Vector to the Activation Derivative Function	Returns a Vector of values, where the Activation Deivative has been applied to them	-	-

3.1.5 Data Logger Tests

Test No.	Test Name	Input Data / Description	Expected Output	Pass / Fail	Testing Evidence
1	Heap Sort Decending	A randomnly generated input list	Sorts the list of items into Descending order	Pass	4.1
2	Add Point	A Data Point matching the data structure of the DataCollector	Point is added to Data Points list	Pass	4.2
3	Match Data Struture with Single	Data Structure contrains an index with a Single-Typed definition	No error thrown	Pass	4.3
4	Match Data Struture with Multi-Typed	Data Structure contrains an index with a Multi-Typed definition	No error thrown	Pass	4.4
5	Match Data Struture with List-Typed	Data Structure contrains an index with a List-Typed definition	No error thrown	Pass	4.5
6	Match Data Structure Error	Try match point with structure which does not match	Error is thrown with correct info	Pass	4.6
7	Select Query	Select from DataLogger with an Index and Search Contents	Returns a list of the selected column where the Search Contents Matches	Pass	4.7
8	Save Data Points	Invoke Save method on DataLogger Object	Saves Data Points to specified File	Pass	4.8
9	Load Data Points	Invoke Load method on DataLogger Object	Loads Data Points from specified File	Pass	4.9

3.1.6 Simulation Tests

Test No.	Test Name	Input Data / Description	Expected Output	Pass / Fail	Testing Evidence
1	Creation of Agent	Run progam as normal	Agent is created as an instance of the Agent Class	Pass	5.1
2	Creation of Enemies	Run program as normal with the "StartEnemyCount" Parameter ≥ 1	Up to the ammount of specified Enemies are created	Pass	5.2
3	Enemies Pathfind towards Agent	Run program as normal with "StartEnemyCount" Parameter ≥ 1	The spawned enemies pathfind towards the agent using the defined pathfinding algorithm	-	-
4	Getting Tile Data	Call .GetTileVec-tor(worldMap, enemyList[]) with arguments for worldMap and the list of current Enemies	Returns a Vector of the surrounding tile objects	Pass	5.4
5	Convert Tile Data	Call .TileVectorPostPro-cess(tileVec) with argument of the result from the Test Above	Converts Tile Data into two vectors, Grayscale Colour and Tile Type	Pass	5.5

6	Reward System Test	-	Expected reward is given to agent	Pass	5.6
7	World Generates to an Acceptable Standard	Run program as normal	Generates 2d Terrain which roughly looks realistic	Pass	5.7
8	World Generation Conforms to Parameters	Utilise inputted parameters to identify the effect they have on the world Generation	Terrain changes depending on inputting Parameters	Pass	5.8
9	Perlin Noise retains Continuity	Generate two worlds with the same seed	Perlin Noise returns same value when using the same seed twice	Pass	5.9

3.2 Testing Evidence

3.2.1 User Input and Program Output Evidence

Evidence 1.1

The .json file which is being loaded

```

{
  "EnterValues": 1,
  "GenerateThreaded": 0,
  "EnableEnemies": 1,
  "SaveWeights": 1,
  "StepDelay": 0,
  "Debug": 0,
  "DebugScale": 1,

  "WorldSize": 64,
  "TileWidth": 8,
  "TileBorder": 0,

  "OctavesTerrain": 7,
  "PersistenceTerrain": 0.6,
  "WorldScale": 3.2,

  "OctavesTrees": 4,
  "PersistenceTrees": 0.95,
  "PoissonKVal": 20,
  "TreeSeedOffset": 1000,
  "TreeHeight": 0.15,
  "InteractableTileBorder": 0,
  "TreeBeachOffset": 0.05,

  "Grayscale": 0,
  "Water": 0.43,
  "Coast": 0.48,
  "Grass": 0.63,
  "Mountain": 1.0,

  "TreeType": "Wood",

  "StartEnemyCount": -13,
  "AgentAttackRange": 1,

  "ColourWater": [18, 89, 144],
  "ColourCoast": [245, 234, 146],
  "ColourGrass": [26, 148, 49],
  "ColourMountain": [136, 140, 141],
  "ColourTree": [13, 92, 28],
  "ColourPlayer": [233, 182, 14],
  "ColourEnemy": [207, 2, 2],

  "MoveReward": 0,
  "CollectItemReward": 0.1,
  "DeathReward": -0.1,
  "ExploreReward": 0.01,
  "AttackReward": 0.5,
  "AttackFailedReward": -0.1,
  "NoopReward": 0,

  "TargetReplaceRate": 5,
  "EREnabled": 1,
  "ERBuffer": 1000,
  "ERSampleRate": 100,
  "ERSampleSize": 10,

  "DeepQLearningLayers": [49, 64, 32, 16, 7],
  "DQLEpoch": 100,
  "DQLearningMaxSteps": 10000,
  "DQLOffset": 3,
  "DQLEpsilon": 0.5,
  "DQLEpsilonRegression": 0.99998,
  "DQLearningRate": 0.75,
  "DQLGamma": 0.8
}

```

Printing the loaded Json File to console to Console to check the values match

```

{'EnterValues': 1, 'GenerateThreaded': 0, 'EnableEnemies': 1, 'SaveWeights': 1, 'StepDelay': 0, 'Debug': 0, 'DebugScale': 1, 'WorldSize': 64, 'TileWidth': 8, 'TileBorder': 0, 'OctavesTerrain': 7, 'PersistenceTerrain': 0.6, 'WorldScale': 3.2, 'OctavesTrees': 4, 'PersistenceTrees': 0.95, 'PoissonKVal': 20, 'TreeSeedOffset': 1000, 'TreeHeight': 0.15, 'InteractableTileBorder': 0, 'TreeBeachOffset': 0.05, 'Grayscale': 0, 'Water': 0.43, 'Coast': 0.48, 'Grass': 0.63, 'Mountain': 1.0, 'TreeType': 'Wood', 'StartEnemyCount': 5, 'AgentAttackRange': 1, 'ColourWater': [18, 89, 144], 'ColourCoast': [245, 234, 146], 'ColourGrass': [26, 148, 49], 'ColourMountain': [136, 140, 141], 'ColourTree': [13, 92, 28], 'ColourPlayer': [233, 182, 14], 'ColourEnemy': [207, 2, 2], 'MoveReward': 0, 'CollectItemReward': 0.1, 'DeathReward': -0.1, 'ExploreReward': 0.01, 'AttackReward': 0.5, 'AttackFailedReward': -0.1, 'NoopReward': 0, 'TargetReplaceRate': 5, 'EREnabled': 1, 'ERBuffer': 1000, 'ERSampleRate': 100, 'ERSampleSize': 10, 'DeepQLearningLayers': [49, 64, 32, 16, 7], 'DQLEpoch': 100, 'DQLearningMaxSteps': 10000, 'DQLOffset': 3, 'DQLEpsilon': 0.5, 'DQLEpsilonRegression': 0.99998, 'DQLearningRate': 0.75, 'DQLGamma': 0.8}

```

Evidence 1.2

Console Output when parameters are within specified ranges

Parameters within Specified Ranges
Created New World - Epoch 205407

A Screenshot of the .json file where the Ranges are defined

```
Parameters > E Range.param
1  {
2    "StepDelay": [0,null],
3
4    "WorldSize": [8,1024],
5    "TileWidth": [1,8],
6    "TileBorder": [0,3],
7
8    "OctavesTerrain": [0,20],
9    "PersistenceTerrain": [0,1],
10   "WorldScale": [0.1,null],
11
12   "OctavesTrees": [0,20],
13   "PersistenceTrees": [0,1],
14   "PoissonRVal": [0,null],
15   "PoissonKVal": [0,null],
16   "TreeHeight": [0,1],
17   "InteractableTileBorder": [0,10],
18   "TreeBeachOffset": [0,1],
19
20   "Grayscale": [0,1],
21   "Water": [0,1],
22   "Coast": [0,1],
23   "Grass": [0,1],
24   "Mountain": [0,1],
25
26   "StartEnemyCount": [0, 100],
27
28   "TargetReplaceRate": [5,300],
29   "ERBuffer": [1000, 10000],
30   "ERSampleRate": [1,100],
31   "ERSampleSize": [10, 1000],
32
33   "DQLearningMaxSteps": [0,null],
34   "DQLOffset": [0,20],
35   "DQLEpsilon": [0,1],
36   "DQLEpsilonRegression": [0,1],
37   "DQLearningRate": [0,1],
38   "DQLGamma": [0,1]
39 }
```

Evidence 1.3

The given out of range parameter - subceeding

"StartEnemyCount": -13,

The specified range it should be within

"StartEnemyCount": [0, 100],

The Exception thrown when the program is run

```
range
Exception: 'StartEnemyCount' of value -13, has subceeded the range: 0-100
PS E:\GithubRepos\CompSciME4>
```

Evidence 1.4

The given out of range parameter - exceeding

"TreeBeachOffset": 1.2,

The specified range it should be within

```
InteractableFileOffset: [
  "TreeBeachOffset": [0,1],
```

The Exception thrown when the program is run

```
cd range
Exception: 'TreeBeachOffset' of value 1.2, has exceeded the range: 0-1
PS F:\Github\Paper\CompSci\5A>
```

Evidence 1.5

The Console prompt if the user wants to load Network Weights

```
Load weights (Y/N): Y
State file name: DQNetwork
```

The file the program is loading

```
▼ DQLearningData ●
  ▢ DQNetwork.dqn M
```

The testing step resumes at 400, underlined in Red

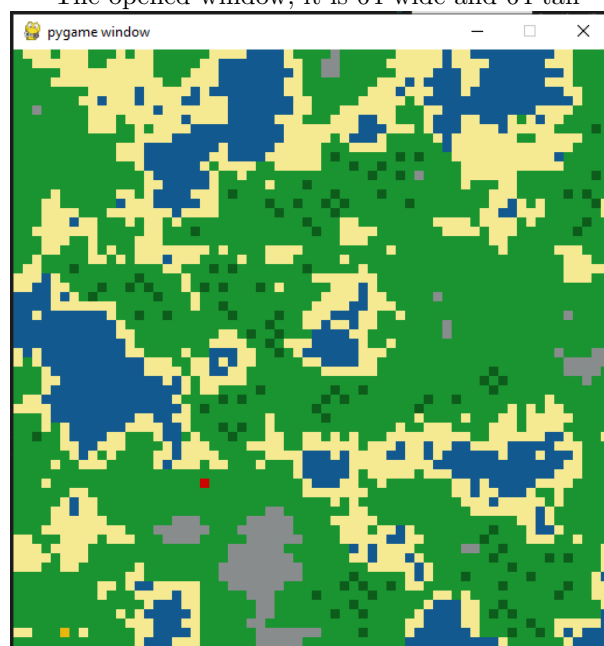
```
Created New World, Seed: 312203
Load weights (Y/N): Y
State file name: DQNetwork
Created New World, Seed: 765802
Created New World, Seed: 274263
Created New World, Seed: 142187
Created New World, Seed: 613313
Created New World, Seed: 961492
Created New World, Seed: 493768
Created New World, Seed: 551641
Created New World, Seed: 133180
400 2.049999999999966 0.49601591773672193
Created New World, Seed: 310069
PS F:\Github\Paper\CompSci\5A>
```

Evidence 1.6

The width/height of the window

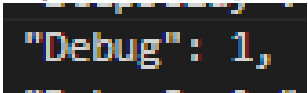
```
"WorldSize": 64,
```

The opened window, it is 64 wide and 64 tall

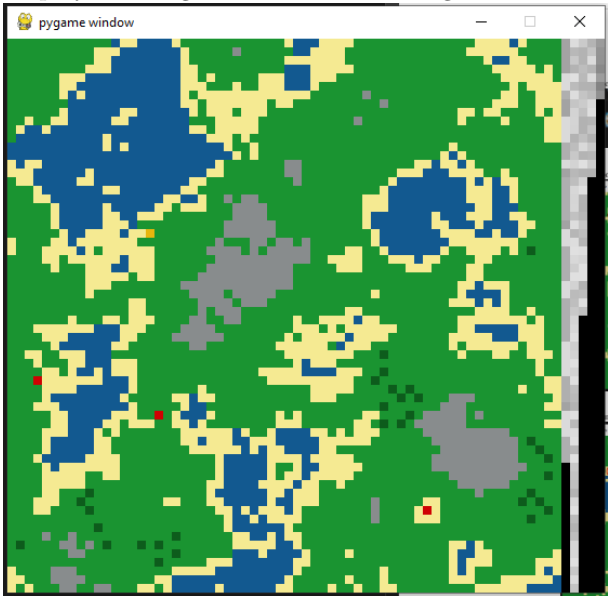


Evidence 1.7

Debug being set to 1 in the parameters file

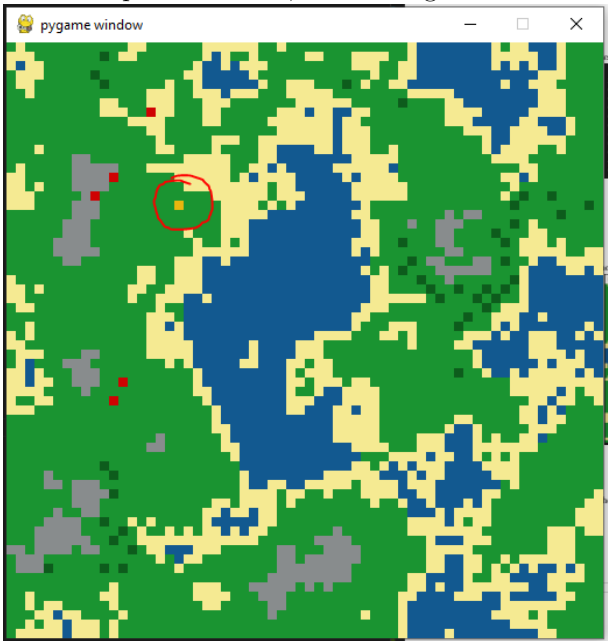


The displayed debug information to the right of the Window



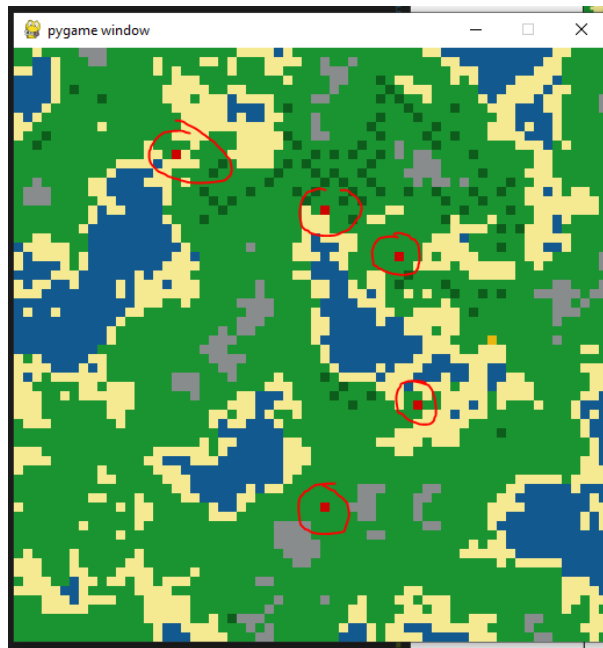
Evidence 1.8

The opened window, with the agent circled



Evidence 1.9

The opened window, with the enemies circled



Evidence 1.10

The correctly displayed console outputs

```
1200 2.089999999999997 0.4881427377231092
Created New World, Seed: 299891
Created New World, Seed: 551234
Created New World, Seed: 419121
Created New World, Seed: 241104
1300 3.5799999999999934 0.4871674181391277
Created New World, Seed: 251077
Created New World, Seed: 479658
Created New World, Seed: 213276
Created New World, Seed: 976354
Created New World, Seed: 774313
Created New World, Seed: 237960
1400 3.539999999999999 0.4861940472644421
Created New World, Seed: 344052
Created New World, Seed: 607949
Created New World, Seed: 102154
Created New World, Seed: 171940
Created New World, Seed: 356413
Created New World, Seed: 50990
Created New World, Seed: 225113
Created New World, Seed: 981988
1500 3.399999999999986 0.4852226212054902
Created New World, Seed: 61676
Created New World, Seed: 9403
Created New World, Seed: 368695
Created New World, Seed: 466339
Created New World, Seed: 851475
Created New World, Seed: 721476
Created New World, Seed: 629285
Created New World, Seed: 664084
Created New World, Seed: 589992
1600 3.1099999999999812 0.4842531360764887
```

3.2.2 Matrix Implementation Tests

Evidence 2.1

Creating a Matrix with a Tuple

```
1 matrix = Matrix((3, 4))
2 print(matrix)
```

The output of the above code:

```
['0', '0', '0', '0']
['0', '0', '0', '0']
['0', '0', '0', '0']
```

Evidence 2.2

Creating a Matrix with a 2d List

```
1 values = [[3, 4],
2           [1, 2]]
3 matrix = Matrix(values)
4 print(matrix)
```

The output of the above code:

```
['3', '4']
['1', '2']
```

Evidence 2.3

Creating a Matrix with a 1d List

```
1 values = [1, 2, 3, 4]
2 matrix = Matrix(values)
3 print(matrix)
```

The output of the above code:

```
['1']
['2']
['3']
['4']
```

Evidence 2.4

Printing a Matrix to the console

```
1 values = [[4, 3],
2           [2, 1]]
3 matrix = Matrix(values)
4 print(matrix)
```

The output of the above code:

```
['4', '3']
['2', '1']
```

Evidence 2.5

Creating a Randomised Matrix

```
1 matrix = Matrix((2, 2), random=True)
2 print(matrix)
```

The output of the above code:

```
['-0.20778786420611217', '-0.1398260152332772']
['0.19471852312767213', '-0.21125677633285878']
```

Evidence 2.6

Creating an Identity Matrix

```
1 matrix = Matrix((3, 3), identity=True)
2 print(matrix)
```

The output of the above code:

```
['1', '0', '0']
['0', '1', '0']
['0', '0', '1']
```

Evidence 2.7

Matrix Addition Calculation

```
1 values = [[4, 3],
2           [2, 1]]
3 matrix = Matrix(values)
4 values2 = [[3, 4],
5            [1, 2]]
6 matrix2 = Matrix(values2)
7
8 result = matrix + matrix2
9 print(result)
```

The output of the above code:

```
['7', '7']
['3', '3']
```

Evidence 2.8

Matrix Subtraction Calculation

```
1 values = [[4, 3],
2           [2, 1]]
3 matrix = Matrix(values)
4 values2 = [[3, 4],
5            [1, 2]]
6 matrix2 = Matrix(values2)
7
8 result = matrix - matrix2
9 print(result)
```


The output of the above code:

```
['1', '-1']  
['1', '-1']
```

Evidence 2.9

Matrix Multiplication Calculation

```
1 values = [[4, 3],  
2           [2, 1]]  
3 matrix = Matrix(values)  
4 values2 = [[3, 4],  
5            [1, 2]]  
6 matrix2 = Matrix(values2)  
7  
8 result = matrix * matrix2  
9 print(result)
```

The output of the above code:

```
['15', '22']  
['7', '10']
```

Evidence 2.10

Matrix Scalar Multiplication Calculation

```
1 values = [[4, 3],  
2           [2, 1]]  
3 matrix = Matrix(values)  
4  
5 result = matrix * 3  
6 print(result)
```

The output of the above code:

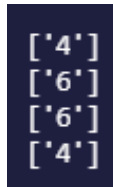
```
['12', '9']  
['6', '3']
```

Evidence 2.11

Vector Hadamard Product Calculation

```
1 values = [1, 2, 3, 4]  
2 vector = Matrix(values)  
3  
4 values = [4, 3, 2, 1]  
5 vector2 = Matrix(values)  
6  
7 result = vector * vector2  
8 print(result)
```

The output of the above code:



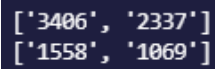
```
['4']
['6']
['6']
['4']
```

Evidence 2.12

Matrix Power Calculation

```
1 values = [[4, 3],
2           [2, 1]]
3 matrix = Matrix(values)
4
5 result = matrix ** 5
6 print(result)
```

The output of the above code:



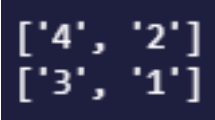
```
['3406', '2337']
['1558', '1069']
```

Evidence 2.13

Matrix Transpose Calculation

```
1 values = [[4, 3],
2           [2, 1]]
3 matrix = Matrix(values)
4
5 result = matrix.Transpose()
6 print(result)
```

The output of the above code:



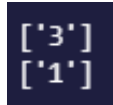
```
['4', '2']
['3', '1']
```

Evidence 2.14

Matrix Select Column

```
1 values = [[4, 3, 6],
2           [2, 1, 5]]
3 matrix = Matrix(values)
4
5 result = matrix.SelectColumn(1)
6 print(result)
```

The output of the above code:



```
['3']
['1']
```

Evidence 2.15

Matrix Select Row

```
1 values = [[4, 3, 6],
2           [2, 1, 5]]
3 matrix = Matrix(values)
4
5 result = matrix.SelectRow(0)
6 print(result)
```

The output of the above code:



```
[4, 3, 6]
```

Evidence 2.16

Vector Max

```
1 values = [4, 3, 6, 1, 2, 5]
2 vector = Matrix(values)
3
4 result = vector.MaxInVector()
5 print(result)
```

The output of the above code:



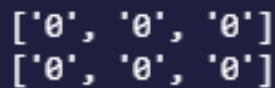
```
(6, 2)
```

Evidence 2.17

Matrix Clear

```
1 values = [[4, 3, 6],
2           [2, 1, 5]]
3 matrix = Matrix(values)
4
5 matrix.Clear()
6 print(matrix)
```

The output of the above code:



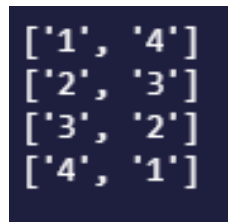
```
['0', '0', '0']
['0', '0', '0']
```

Evidence 2.18

Matrix Combine Vectors

```
1 values = [1, 2, 3, 4]
2 vector = Matrix(values)
3
4 values = [4, 3, 2, 1]
5 vector2 = Matrix(values)
6
7 vectorList = [vector, vector2]
8
9 result = Matrix.CombineVectorsHor(vectorList)
10 print(result)
```

The output of the above code:



Evidence 2.19

Matrix Sum

```

1 values = [[4, 3, 6],
2           [2, 1, 5]]
3 matrix = Matrix(values)
4
5 result = matrix.Sum()
6 print(result)

```

The output of the above code:



Evidence 2.20

Console Output, all Tests have passed with no failures

```

10000/10000 | CreateVectorFrom1DList
10000/10000 | CreateMatrixFrom2DList
10000/10000 | CreateMatrixFromTuple
10000/10000 | CreateIdentityMatrix

```

Evidence 2.21

Console Output, all Tests have passed with no failures

```

10000/10000 | NoMatchingInitCase
10000/10000 | UnableToCreateIdentityMat

```

Evidence 2.22

Console Output, all Tests have passed with no failures

```

10000/10000 | AdditionMatrix
10000/10000 | AdditionInteger
10000/10000 | SubtractionMatrix
10000/10000 | SubtractionInteger
10000/10000 | MultiplicationInteger
10000/10000 | MultiplicationHadamardVector
10000/10000 | MultiplicationMatrix
10000/10000 | Power
10000/10000 | Transpose
10000/10000 | SelectColumn
10000/10000 | SelectRow
10000/10000 | CombineVectorHorizontal
10000/10000 | Sum
10000/10000 | MaxInVector
10000/10000 | Clear

```

Evidence 2.23

Console Output, all Tests have passed with no failures

```
10000/10000 | NotOfTypeVector
10000/10000 | VectorsNotOfSameLength
10000/10000 | NoMatchingMultiplyCase
10000/10000 | NoMatchingAdditionCase
10000/10000 | NoMatchingSubtractionCase
10000/10000 | NoMatchingPowerCase
10000/10000 | MismatchOrdersAdd
10000/10000 | MismatchOrdersSub
10000/10000 | MismatchOrdersMul
10000/10000 | SumOfMatrixReqNumericalVals
10000/10000 | ColumnOutOfRange
10000/10000 | ColumnMustBeInteger
10000/10000 | RowOutOfRange
10000/10000 | RowMustBeInteger
```

3.2.3 Deep Reinforcement Learning Algorithm Evidence

Evidence 3.1

The Neural Network objects in Memory

```
Load weights (Y/N): n
MainNetwork: <deepqlearning.NeuralNet object at 0x000001FCE8C63D00>
TargetNetwork: <deepqlearning.NeuralNet object at 0x000001FCE8D17A00>
```

Evidence 3.2

The layer sizes upon creating the Networks

```
Load weights (Y/N): n
Layer Size: 5
Layer Size: 5
```

The list of layer sizes in the parameters file

```
"DeepQLearningLayers" : [49, 64, 32, 16, 7],
```

Evidence 3.3

Evidence 3.4

Evidence 3.5

Evidence 3.6

Evidence 3.7

Evidence 3.8

Pushing items to the front of the Double Ended Queue

```
1 deque = Deque(10)
2 deque.PushFront(3)
3 print("Added 3:", deque.queue)
4 deque.PushFront(-5)
5 print("Added -1:", deque.queue)
6 deque.PushFront(9)
7 print("Added 9:", deque.queue)
```

The output of the above code:

```
Added 3: [3, None, None, None, None, None, None, None, None, None]
Added -1: [3, -5, None, None, None, None, None, None, None, None]
Added 9: [3, -5, 9, None, None, None, None, None, None, None]
```

Evidence 3.9

Creating a Double Ended Queue with a length of 4, add Push Items to it, and get the Items in First and Last

```
1 deque = Deque(4)
2 deque.PushFront(3)
3 deque.PushFront(-5)
4 deque.PushFront(9)
5 deque.PushFront(4)
6 deque.PushFront(-4)
7
8 print("First:", deque.First())
9 print("Last:", deque.Last())
10 print("Queue:", deque.queue)
```

The output of the above code:

```
First: -4
Last: -5
Queue: [-4, -5, 9, 4]
```

Evidence 3.10

Create a Double Ended Queue and Sample items from the Queue

```
1 deque = Deque(4)
2 deque.PushFront(3)
3 deque.PushFront(-5)
4 deque.PushFront(9)
5 deque.PushFront(4)
6 deque.PushFront(-4)
7
8 print("Sample 1:", deque.Sample(2))
9 print("Sample 2:", deque.Sample(2))
10 print(deque.queue)
```

The output of the above code:

```
Sample 1: [-5, 4]
Sample 2: [-5, 9]
[-4, -5, 9, 4]
```

3.2.4 Data Logger Evidence

Evidence 4.1

Randomly Generated Unsorted List, sorted by the 1st Element to form the Sorted List

```

1  inputList = [[random.randint(-10,10), random.randint(-10,10)] for i in range(5)]
2  print("Unsorted List:")
3  for item in inputList:
4  print(item)
5
6  dl = DataCollector("SortingTest", [int, int], False)
7
8  dl.LogDataPointBatch(inputList)
9
10 sortedList = dl.HeapSort(0)
11
12 print("Sorted List:")
13 for item in sortedList:
14 print(item)

```

The output of the above code:

```

Unsorted List:
[0, 6]
[-6, -4]
[-3, -2]
[-2, 1]
[7, -1]
Sorted List:
[7, -1]
[0, 6]
[-2, 1]
[-3, -2]
[-6, -4]

```

Evidence 4.2

Adding a single point: [5, 2] to DataLogger

```

1  dl = DataCollector("AddPointTest", [int, int], False)
2  print("Before: ", dl.dataPoints)
3
4  dl.LogDataPoint([5, 2])
5
6  print("After: ", dl.dataPoints)

```

The output of the above code:

```

Before:  []
After:  [[5, 2]]

```

Evidence 4.3

Test Data Point matches struture

```

1  dl = DataCollector("Match Single Types", [int, float], False)
2
3  print("Matches Structure: ", dl.CheckMatchStructure([-3, 2.2]))

```

The output of the above code:

```
Matches Structure: True
```

Evidence 4.4

Test Data Point matches structure

```
1 dl = DataCollector("Match Multi Typed", [bool, [float, int]], False)
2
3 print("Matches Structure: ", dl.CheckMatchStructure([False, 4.5]))
4 print("Matches Structure: ", dl.CheckMatchStructure([True, -9]))
```

The output of the above code:

```
Matches Structure: True
Matches Structure: True
```

Evidence 4.5

Test Data Point matches structure

```
1 dl = DataCollector("Match List Type", [bool, str], False)
2
3 print("Matches Structure: ", dl.CheckMatchStructure([True, ["Matt", "Isabel", "Tristan", "Chris"]]))
```

The output of the above code:

```
Matches Structure: True
```

Evidence 4.6

Test error thrown when Data Point doesnt match the given structure

```
1 try:
2 dl = DataCollector("Match Data Structure Error", [str, int], False)
3
4 print("Matches Structure: ", dl.CheckMatchStructure(["Steve Preston", True]))
5 except Exception as x:
6 print(x)
```

The output of the above code:

```
Type: <class 'bool'> != Data Structure Type: <class 'int'>
[<class 'str'>, <class 'int'>]
```

Evidence 4.7

Select Prime numbers in 1st index

```
1 inputList = [[random.randint(-10,10), random.randint(-10,10)] for i in range(5)]
2 print("Random List:")
3 for item in inputList:
4 print(item)
5
6 dl = DataCollector("Select List", [int, int], False)
7
8 dl.LogDataPointBatch(inputList)
9
10 sortedList = dl.Select(0, [1,2,3,5,7])
11
```



```

12 | print("Selected List:")
13 | for item in sortedList:
14 |     print(item)

```

The output of the above code:

```

Random List:
[9, -5]
[8, 3]
[1, -8]
[-1, 4]
[4, -10]
Selected List:
[1, -8]

```

Evidence 4.8

Test for saving a file

```

1 | inputList = [[random.randint(-10,10), random.randint(-10,10)] for i in range(5)]
2 | print("Saved List:")
3 | for item in inputList:
4 |     print(item)
5 |
6 | dl = DataCollector("Save-Load Test", [int, int], False)
7 |
8 | dl.LogDataPointBatch(inputList)
9 |
10 | dl.SaveDataPoints()

```

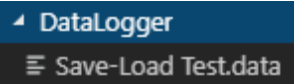
The saved Data Points

```

Saved List:
[8, 10]
[-7, -1]
[-1, -7]
[4, 1]
[5, -6]

```

The saved file "Save-Load Test.data"



Evidence 4.9

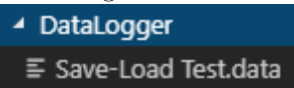
Test for loading a file

```

1 | dl = DataCollector("Save-Load Test", [int, int], True)
2 |
3 | print("Loaded List:")
4 | for item in dl.dataPoints:
5 |     print(item)

```

The File we're loading from "Save-Load Test.data"



The loaded Data Points

```
Loaded List:
[8, 10]
[-7, -1]
[-1, -7]
[4, 1]
[5, -6]
```

3.2.5 Simulation Evidence

Evidence 5.1

The Agent Object stored in Memory

```
<newAgent.Agent object at 0x0000015315DA1B80>
```

Evidence 5.2

The Enemy Objects stored in Memory

```
[<enemy.Enemy object at 0x0000026F669D0100>, <enemy.Enemy object at 0x0000026F66AB19D0>, <enemy.Enemy object at 0x0000026F66AB1A30>, <enemy.Enemy object at 0x0000026F66AB1A90>, <enemy.Enemy object at 0x0000026F66AB1AF0>]
```

Evidence 5.3

Video Evidence

Evidence 5.4

Tile Data Objects are returned in a Vector

```
['<worldClass.Tile object at 0x0000020C7F72B880>']
['<worldClass.Tile object at 0x0000020C7F72F4C0>']
['<worldClass.Tile object at 0x0000020C7F735100>']
['<worldClass.Tile object at 0x0000020C7F735D00>']
['<worldClass.Tile object at 0x0000020C7F738940>']
['<worldClass.Tile object at 0x0000020C7F73C580>']
['<worldClass.Tile object at 0x0000020C7F7411C0>']
['<worldClass.Tile object at 0x0000020C7F72B880>']
['<worldClass.Tile object at 0x0000020C7F72F4F0>']
```

Evidence 5.5

Grayscale Values in a Vector

```
['0.39308235294117644']
['0.39308235294117644']
['0.39308235294117644']
['0.39308235294117644']
['0.39308235294117644']
['0.39308235294117644']
['0.39308235294117644']
['0.39308235294117644']
['0.8912039215686275']
['0.8912039215686275']
['0.39308235294117644']
```

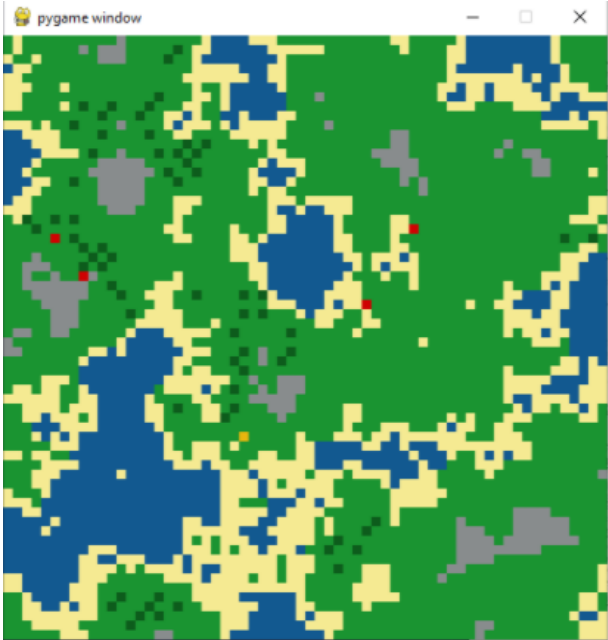
Evidence 5.6

Reward on Left, and the chosen action on Right

-0.1 5

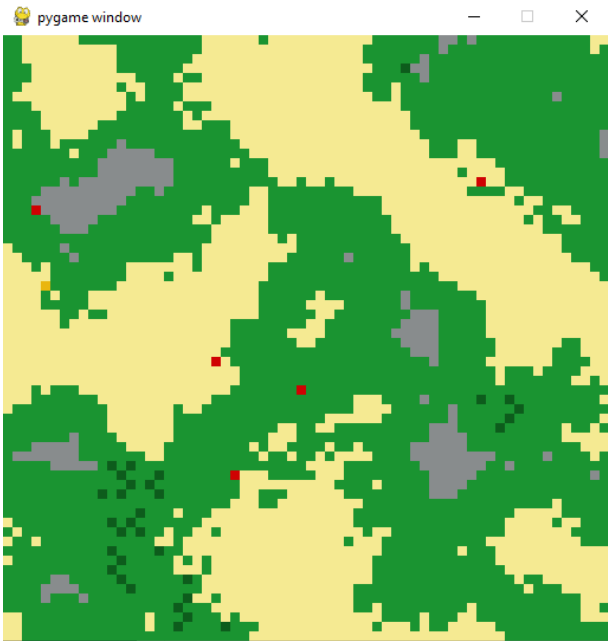
Evidence 5.7

World Generated, is of an Acceptable Standard



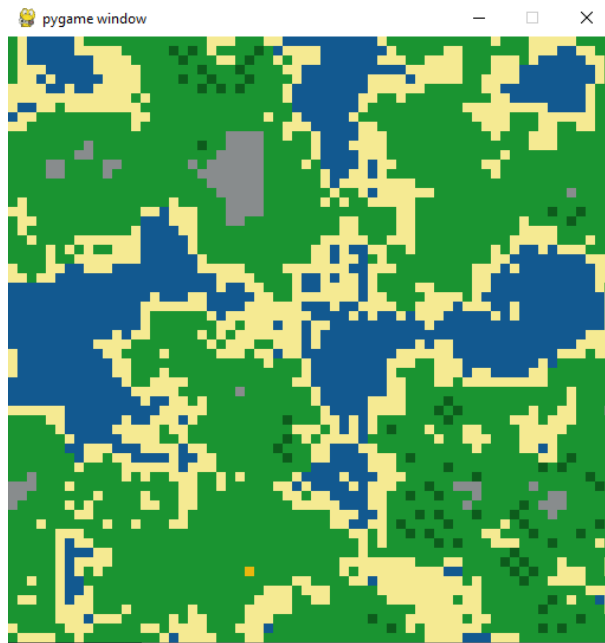
Evidence 5.8

Changed Water Value creates no Water and more Beach

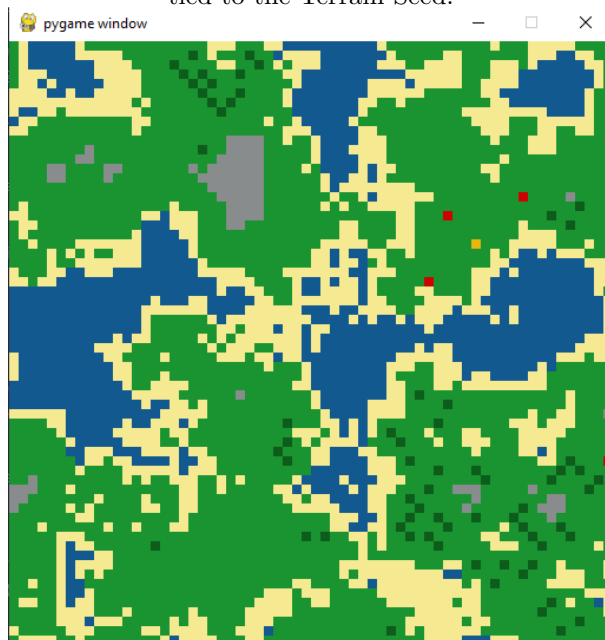


Evidence 5.9

Generated with seed 420



Generated again with the seed 420. Note different Trees, Enemy and Agent Positions, due to them not being tied to the Terrain Seed.



4 Evaluation

4.1 Evaluation of Objectives

In this section, I will evaluate my overarching objectives I set out to complete.

4.1.1 Reading user inputted data

The user can input the parameters through a json file, and these parameters are checked against a range file to check they are within the specified size. All of the parameters are read correctly and utilised within the Program.

The Machine Learning Data is read from .dqn files. The Learning is resumed from where it was saved from with all the Weights and Biases intact.

4.1.2 Generating the Environment

At the start of the program an instance of World Class is created and the Generate methods are invoked. These methods utilise Perlin Noise and Poisson Disc Sampling. The Terrain values are stored in a 2d list of Tile Objects which store the Height, Type and Colour data for each Tile. The Poisson Disc Sampling Generates a list of points which Trees are then generated at those positions. The Width of the world and Tile colours are determined by the Input Parameters.

4.1.3 Displaying the world to a Pygame Window

Upon generating the Map Data the Terrain is displayed in a grid to the Pygame Window, it is represented as a grid of tiles of the pixel width loaded in by the Inputted Parameters. The Agent and Enemies are Drawn at their according positions, taking up entire Tile. If Debug mode is enabled, a representation of the Neural Network will be displayed on the right hand side of the window.

4.1.4 Simple Agent with a set of Actions

An Agent can be created as an object and works along side the Dual Neural Network Object to enable interactions between the environment and the Network. The Agent can collect the surrounding Tile Data using the **GetTileVector** Method, this can then be converted into the Networks Input Vector using the **TileVectorPostProcess** Method. There exists Methods to Take a given Action, normally outputted by the Network. Along with Methods to Calculate Reward for an Action given a State, or the Maximum Possible Reward Given a State.

There also exists Methods to Reset the Agent to its default values. Along with Determining the Agents Spawn Position when given a WorldMap Object.

4.1.5 Matrix Class with Standard Operations

A Matrix can be created using 3 different methods. First using a Tuple of Integers, a new Matrix will be created of that size, with initialised 0 values. Second using a preexisting 2d list

of values, a new Matrix will be created with these dimensions and values. Thirdly a 1d list of values can be used to create a 1 wide Vector of values, where it reads each value into the 1st position of each row.

All standard operations for the Matrix Object are implemented using Operator Overloading to make code less bloated. All are written efficiently utilising minimum complexity algorithms. Addition can be carried out utilising the + Operator. Subtraction can be carried out utilising the – Operator. Multiplication and Scalar Multiplication are both carried out utilising the * Operator. Power Operation is carried out utilising the ^ operator. A Matrix can be converted to a Formatted String implicitly by using it in a string context.

All Matrice Operations have appropriate Exceptions with descriptive Error Messages. They will throw errors when incorrect Data is provided to the specified Operation.

4.1.6 Creation of a Reinforcement Learning Model

A Dual Neural Network can be created as an object, which stores two Neural Network Objects, Main and Target. The Dual Neural Network contains the Primary Method **Step** which invokes a Series of Lower Level Methods to perform a singular Time Step. The Neural Network Object store a List of Layers Objects which are dynamically created from the Input Parameters. Each Layer contains a Weight Matrix, Bias Vector, and Output Vector. The Lowest Level methods for Forward and Back Propagation are contained within the Layer Object.

First Forward Propagation occurs on the Main and Target Network. Then results of the Main Network are taken to choose the action for the Agent. Epsilon Greedy is implemented to determine whether to choose the random or predicted result. This Action is then fed to the Agent, along with calculating the reward for that Action. The Loss of the Main Network is then calculated using a modified Bellman Equation for Dual Neural Networks. This Loss is used for Back Propagating the Main Neural Network. The Main Networks Weights are copied to the Target Network every specified ammount of steps. Every specified ammount of steps, Experience Replay is performed to learn from past experiences again.

The combination of these steps form a functional Dual Neural Network utilising a Reinforcement Learning Model.

4.1.7 Creation of a Data Logger

A Data Logger Class can be used to Log and Store Data Points at various parts of the Program. Each Data Point is stored as a Tuple of Values as part of a .data file. These files are stored as Binary Files, and are Read into the Program upon launch.

As part of the Data Logger you can sort points utilising a Heap Sort to sort through Data.

4.2 Answering the Proposed Question

As part of my Machine Learning Investigation I proposed the Question:

Can I develop a Machine Learning Algorithm to survive in a pseudorandom, open-world environment?

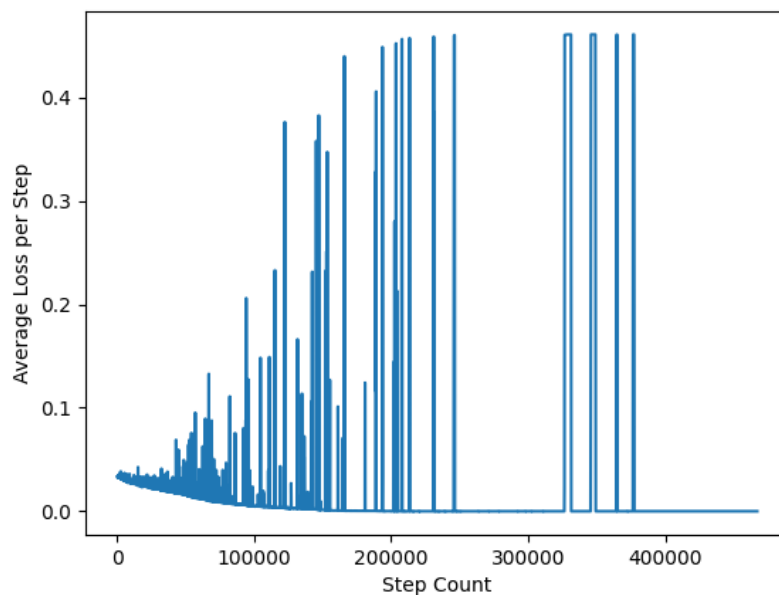
I aimed to answer this question by designing and creating a Deep Reinforcement Learning Model utilising a Deep Neural Network, along with designing a Simple Simulation for a Machine Learning Agent to survive in.

With the Machine Learning Model I implemented, the Agent was unable to solve the simulation. After being trained to 500000 steps, the accuracy of the Network Improves in comparison to randomly taking actions, but fails to reach higher levels. I attempted to train the Network with varying parameters and Network Structures, but this yielded similar results. The baseline Loss is minimised, but the loss still peaks back to the original value on a regular basis.

This follows suit with one of the Existing Investigations I researched. Crafter only manages to achieve 10% accuracy with the best algorithm, the rest achieving below 5%. In comparison to Human Experts which achieved near 60% accuracy, this is statistically little.

4.3 Analysis of Training Data

I found that the Network is sensitive to its reward structure and Network architecture. When the Reward Structure has an action which gains 0 but also loses 0 reward, the Back Propagation will minimise the Network into purely taking this action. An example of this is when "Noop" or the Null Action is enabled. This ended up in the graph just flatlining towards 0 average loss, where the Network only took the null action 99% of the time.

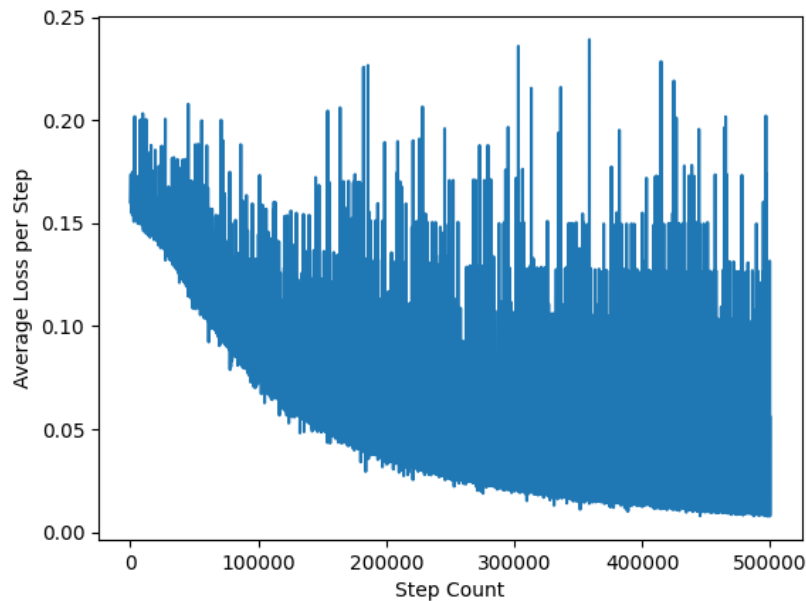


Neural Network flatlining towards 0 loss by only picking "Noop"

Large network architecture with 49 Input Nodes

Enemies Disabled

I am unsure as to what the spikes are, I believe it is due to instabilities in the training architecture. Following on from this failed attempt to train the Network, I removed Noop from the action set. This led to overall weird results, the baseline Loss trends down, but doesn't manage to overall minimise it. I believe this is a sign that the simulation is too complex for the Network architecture to solve.

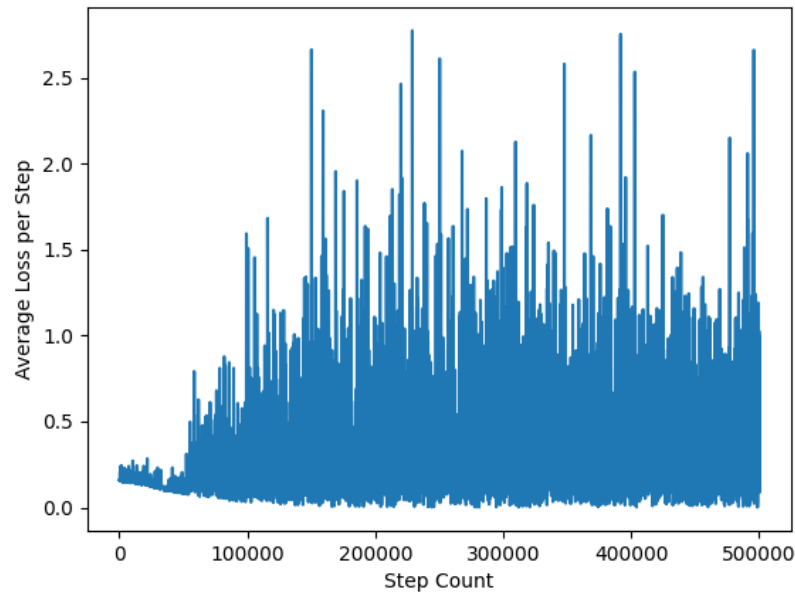


Neural Network attempts to minimise network but fails to solve the simulation

Large network architecture with 49 Input Nodes

Enemies Disabled, Attack and Noop action disabled

I then enabled the enemies with the same Network architecture, this led to different results. The Network clearly places a significance on their existence, but fails to overcome them as a problem. I observed during this training session that the Agent does manage to kill enemies sometimes. but fails to do this consistently. I believe this might be due to the high sensory input.

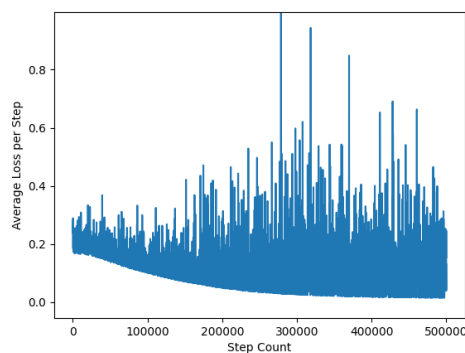


Neural Network struggles with Enemies

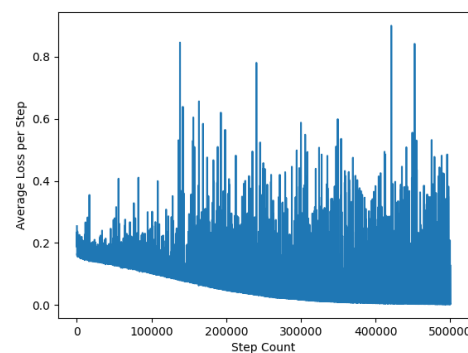
Large network architecture with 49 Input Nodes

Noop action disabled

I also attempted training using different Network architectures, this led to much better results compared to the previous training session with 49 Inputs. This as stated previously may be due to the high sensory input of a larger Network. I think the 25 Input Network performs ever so slightly better than the 9 Input, but this may only be due to random chance.

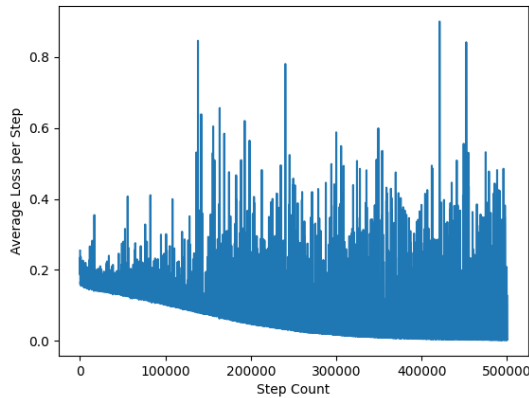


(a) 25 Input Nodes

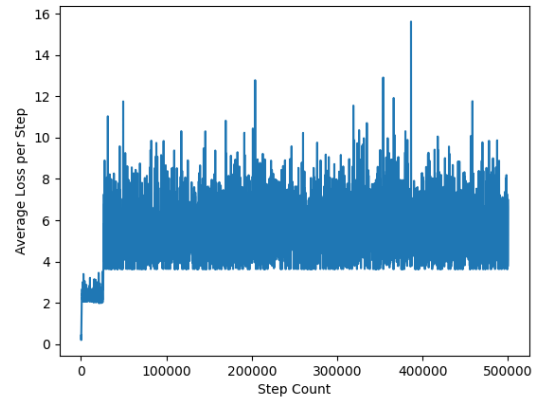


(b) 9 Input Nodes

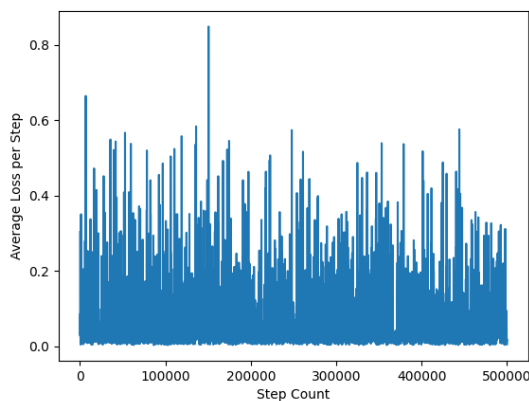
I then chose the best performing Network out of the 3 I tested, which has 25 Inputs, and tried it with all the Activation Functions I've implemented. Previously I had just been using the standard Sigmoid Activation function. This is an attempt to find the best possible Network \Rightarrow Activation Combination.



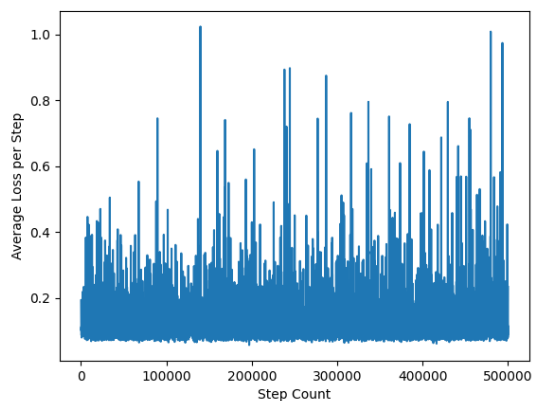
(c) Sigmoid Activation Function



(d) TanH Activation Function



(e) ReLu Activation Function



(f) Leaky ReLu Activation Function

As shown above Sigmoid is clearly the best Activation Function to use for the problem. TanH exhibits weird behaviour which I can't explain. ReLu and Leaky ReLu perform similarly, with Leaky ReLu being slightly better but both may as well be random.

Leaving us with the best Network Architecture with a layer structure of [25, 32, 16, 8, 6], utilising the Sigmoid Activation Function.

4.4 Expert Feedback

I went back to my Expert Shaun in order to collect feedback on my finalised Technical Solution. I asked him a few Questions about my project, paraphrased where necessary.

1. What do you think of the Program?

"Overall I think your project is incredibly visually interesting to look at, I could stare at the graphical output for hours just rooting for the Agent to better itself and kill the Generated Enemies. The User Inputted Parameters are easy to change through the json file, and it is helpful that they are locked between certain ranges to stop the User from crashing their Pc from allocating too much memory. The Terrain generation looks pretty good for just a 4 coloured map generated from Perlin Noise. The Neural Network works as intended, although it's a shame that the Machine Learning Model isn't advanced enough to 'Solve' the Simulation you've designed."

2. Does my Technical Solution achieve all of the Set Goals and Objectives?

"The Program achieves all of the objectives you set out to complete, and it is clear a lot of hard work went into completing your project. Lots of research needs to be carried out in order to understand the complexity behind Reinforcement Learning and all of its individual parts. Debugging this process also becomes increasingly difficult, due to the complex calculations, this demonstrates you have the ability to solve problems independently.

You've also implemented an entire simulation on top of the Dual Neural Network. Which uses more complex algorithms, this demonstrates you can develop multiple Vertical Slices of a project, and intertwine them together in order to create one bigger project. This takes planning skill and a good understanding of OOP in order to pull off."

3. What Criticisms/Improvements would you suggest?

"Considering the scope of the project, you've carried out your completion of this task very well. The only suggestion I would have is to implement a Convolution, which might solve your Training Accuracy Problems. Otherwise a Description of your Project could be printed to console when the main file is run, or a 'ReadMe' text file included in the project files would be useful to any users who have little to no experience with Reinforcement Learning."

4.5 Evaluation of Expert Feedback

I'm glad that my Expert likes my project. After putting so much work into it that is a relief. I think that his suggestions are valid, and in the future I might develop my project further to add a Convolution. This will hopefully boost the accuracy of my Network so I can achieve better training results. The ReadMe text file would also be a good addition, if I was to ever show my project publically.

Shaun has been a great use to me, such as helping me "Sanity Check" myself when my Back Propagation didn't work right off the bat. This help was incredibly valuable in completing my Technical Solution.

4.6 System Improvements

Overall I am happy with my Technical Solution. I achieved all the objectives I set out to complete in my Analysis. I have definitely achieved my primary goal of gaining a deeper understanding about the Maths and Logic behind how Neural Networks work. This has given me a Window into the field of Machine Learning and Artificial Intelligence, which I intend to pursue as part of my later Studies. If I were to complete my NEA again I would apply Machine Learning to a different sector of problem, because Reinforcement Learning has been a tough challenge. It has been kind of disappointing as well that the Network has been failed to solve the simulation I built.

The Improvements I would like to make to my Technical Solution are:

- The Implementation of a Convolutional Neural Network was something I came across in my Initial Research and was mentioned by my Expert. Convolution carries out Pre-Processing on the inputted data before it is even touched by the Neural Network. This in theory would increase the training accuracy of my Network leading to better Results.

- The Optimisation of my Matrix Class by compiling it into *C* through the use of Cython would help speed up the training of the Neural Network. Due to Python being an interpreted language it is comparatively slow compared to the other programming languages I considered using. *C* is a compiled language so it is comparatively a lot faster, about 45 times faster according to some sources online. This could provide an easy way to optimise my Program without having to convert my entire Codebase into a different Language. Although I wish I had used a different language for my Technical Solution, I think Rust would've been the correct choice for this project.
- An increase in complexity of my simulation would provide a greater challenge towards my Agent and Neural Network. I could add a basic crafting system to convert the collected Wood into a sword, or a Hunger Bar so the Agent has to collect food and water in order to survive. I feel as though the Network wouldn't be able to solve these problems effectively though without the implementation of my first improvement, a Convolutional Neural Network.

5 Technical Solution

5.1 main.py

```
1  import pygame
2  from simulation import *
3  import time
4
5  params = Simulation.LoadParameters("Default") # Loads parameters
6  Simulation.CheckParameters(params, "Range") # Checks parameters
7
8  gameSim = Simulation(params) # Create and initiate simulation
9  gameSim.InitiateSimulation()
10
11 # Creates pygame window - includes side debug offset if needed
12 worldResolution = params["WorldSize"] * params["TileWidth"]
13 if params["Debug"]:
14     debugOffset = (len(params["DeepQLearningLayers"]) * params["TileWidth"] * params["DebugScale"])
15 else:
16     debugOffset = 0
17 window = pygame.display.set_mode((worldResolution + debugOffset, worldResolution))
18
19 stepDelay = params["StepDelay"] # Time step Delay
20
21 # Constant loop running
22 running = True
23 while running:
24     for event in pygame.event.get():
25         if event.type == pygame.QUIT: # If window exit than close end program
26             running = False
27
28         if event.type == pygame.KEYDOWN: # Key Down
29             if event.key == pygame.K_F1: # Force Create new world
30                 gameSim.CreateWorld()
31             if event.key == pygame.K_F2: # Force Kill agent
32                 gameSim.agent.alive = False
33
34         if gameSim.step > params["DQLearningMaxSteps"]:
35             running = False
36
37     gameSim.TimeStep() # Perform a timestep
38     time.sleep(stepDelay) # Sleep if needed
39
40     gameSim.RenderToCanvas(window) # Draw to canvas
41
42     pygame.display.update() # Update pygame window to display content
```

5.2 simulation.py

```
1  from worldClass import *
2  from newAgent import *
3  from enemy import *
4  from deepqlearning import *
```

```

5  import random, pygame, math
6
7  # Interface class between Main and Every other class
8  class Simulation():
9      def __init__(self, params): # Constructor for Simulation
10         self.paramDictionary = params
11
12         self.worldMap = None
13         self.network = None
14         self.agent = None
15
16         self.enemyList = []
17
18         self.step = 0
19
20     # Step forward network methods
21     def TimeStep(self): # Steps forward 1 cycle
22         if not self.agent.alive: # Resets Sim if Agent is dead
23             self.ResetOnDeath()
24
25         self.network.TakeStep(self.agent, self.worldMap, self.enemyList) # Take step with Deep Q Network
26
27         if self.paramDictionary["EnableEnemies"]: # If enemies enabled then update enemies
28             self.UpdateEnemies()
29
30         self.step += 1
31
32     def UpdateEnemies(self): # Updates Enemies
33         self.enemyList = [x for x in self.enemyList if x is not None] # Clears None type from list
34
35         for i in range(len(self.enemyList)): # Commits each Enemies actions and sets to None if they died in that
36             self.enemyList[i].CommitAction(self.agent, self.worldMap)
37
38             if not self.enemyList[i].alive: # Removes dead enemies from list
39                 self.enemyList[i] = None
40
41         self.enemyList = [x for x in self.enemyList if x is not None] # Clears None type from list
42
43     # Creation and Initialisation Methods
44     def InitiateSimulation(self): # Initialises Simulation
45         self.CreateWorld()
46         self.CreateAgent()
47
48         self.CreateDeepQNetwork()
49
50     def CreateWorld(self, seed = 0): # Creates new world with specified or random seed
51         if seed == 0: seed = random.randint(0, 999999)
52
53         if self.worldMap == None: # Creates a new world map if one does not exist - otherwise resets the seed
54             self.worldMap = WorldMap(seed, self.paramDictionary)
55         else:
56             self.worldMap.MAP_SEED = seed
57
58         if self.paramDictionary["GenerateThreaded"]: # Generates Terrain using 4 threads if specified
59             self.worldMap.GenerateThreadedParent()

```

```

60         else:
61             self.worldMap.GenerateMap()
62
63         self.worldMap.GenerateTreeArea() # Generates Tree Area
64
65         self.worldMap.RenderMap() # Renders Map and Renders Interactables
66         self.worldMap.RenderInteractables()
67
68         if self.paramDictionary["EnableEnemies"]: # Spawns Enemies if specified
69             self.SpawnEnemies()
70
71         print("Created New World, Seed: {}".format(seed))
72
73     def CreateDeepQNetwork(self, layers = None): # Creates a Deep Q Network with the given Hyper Parameters
74         if layers == None:
75             layers = self.paramDictionary["DeepQLearningLayers"]
76
77         if self.network == None: # Creates a Network if one doesnt already exist
78             if self.paramDictionary["EnterValues"]:
79                 load = input("Load weights (Y/N): ")
80                 if load.upper() == "Y":
81                     fName = input("State file name: ")
82
83                     self.network = DoubleNeuralNet(layers, self.paramDictionary, load=True, loadName=fName)
84             else:
85                 self.network = DoubleNeuralNet(layers, self.paramDictionary)
86         else:
87             self.network = DoubleNeuralNet(layers, self.paramDictionary)
88
89     def CreateAgent(self): # Creates an agent / Resets existing agent
90         if self.agent == None:
91             self.agent = Agent(Agent.SpawnPosition(self.worldMap), self.paramDictionary)
92         else:
93             self.agent.Reset(self.worldMap)
94
95     def SpawnEnemies(self, n = 0): # Spawns <= n enemies on call
96         if n == 0: n = self.paramDictionary["StartEnemyCount"]
97
98         for count in range(n): # Spawns enemies for count
99             spawnLoc = Enemy.SpawnPosition(self.worldMap, self.enemyList)
100             if spawnLoc == None:
101                 continue
102             else:
103                 tempEnemy = Enemy(spawnLoc, self.paramDictionary)
104                 self.enemyList.append(tempEnemy)
105
106     def ResetOnDeath(self): # Resets Simulation if Agent Dies
107         self.CreateWorld()
108         self.CreateAgent()
109         self.enemyList = []
110
111         if self.paramDictionary["EnableEnemies"]: # Spawns Enemies if specified
112             self.SpawnEnemies()
113
114     # Render Methods

```

```

115     def RenderToCanvas(self, window): # Render Content to Canvas
116         TW = self.paramDictionary["TileWidth"]
117         DS = self.paramDictionary["DebugScale"]
118
119         if self.paramDictionary["Debug"]: # Renders debug info for Neural Network if specified
120             for i in range(len(self.network.MainNetwork.layers)):
121                 for k in range(self.network.MainNetwork.layers[i].activations.order[0]):
122                     value = self.network.MainNetwork.layers[i].activations.matrixVals[k][0]
123                     newVal = (math.tanh(value) + 1) / 2
124                     colourTuple = (255 * newVal, 255 * newVal, 255 * newVal)
125
126                     try: # Exceps if colour value out of range
127                         pygame.draw.rect(window, colourTuple, ((self.paramDictionary["WorldSize"] * TW + i * TW *
128                     except:
129                         print(newVal)
130
131         self.worldMap.DrawMap(window) # Draws Content to window
132
133         for i in range(len(self.enemyList)): # Draws enemies to window
134             pygame.draw.rect(window, self.paramDictionary["ColourEnemy"], ((self.enemyList[i].location[0] * TW),
135
136         # Draws Player to window
137         pygame.draw.rect(window, self.paramDictionary["ColourPlayer"], ((self.agent.location[0] * TW), (self.agent
138
139 # Miscellaneous Methods
140
141     @staticmethod
142     def LoadParameters(fname): # Load Parameters from file and store them in a dictionary
143         file = open("Parameters\\{}.param".format(fname), "r")
144         params = json.loads(file.read())
145         file.close()
146         return params
147
148     @staticmethod
149     def CheckParameters(params, fname): # Checks every parameter against the range.parm file
150         file = open("Parameters\\{}.param".format(fname), "r") # Read range file
151         paramRanges = json.loads(file.read()) # Load with json module
152         file.close()
153
154         for param in params: # Checks if parameter is specified in range file - If specified than check against g
155             if param in paramRanges:
156                 valRange = paramRanges[param]
157                 val = params[param]
158
159                 if valRange[1] == None: pass
160                 elif val > valRange[1]:
161                     raise Exception("'{}' of value {}, has exceeded the range: {}-{}".format(param, val, valRange
162
163                 if valRange[1] == None: pass
164                 elif val < valRange[0]:
165                     raise Exception("'{}' of value {}, has subceeded the range: {}-{}".format(param, val, valRang
166
167         print("Parameters within Specified Ranges")

```


5.3 newAgent.py

```

1  from worldClass import *
2  from random import shuffle
3  from matrix import Matrix
4  from copy import copy
5
6  class Agent():
7      def __init__(self, location, params):
8          self.paramDictionary = params
9
10         self.location = location
11
12         self.alive = True
13
14         self.emptyInventory = {"Wood": 0}
15         self.inventory = self.emptyInventory
16
17     # Methods for tile vectors
18     def GetTileVector(self, worldMap, enemyList): # Returns a Vector of Tile Datatype
19         offset = self.paramDictionary["DQLOffset"]
20         sideLength = 2 * offset + 1
21         tileVec = Matrix((sideLength * sideLength, 1))
22
23         blankOceanTile = Tile()
24         blankOceanTile.InitValues(0, 0, self.paramDictionary["ColourWater"]) # Blank ocean tile for edge case
25
26         enemyLocList = [enemyList[i].location for i in range(len(enemyList)) if enemyList[i] is not None]
27
28         n = 0
29         for y in range(self.location[1] - offset, self.location[1] + offset + 1): # Loop through Tiles in surround
30             for x in range(self.location[0] - offset, self.location[0] + offset + 1):
31                 if 0 <= x and x <= self.paramDictionary["WorldSize"] - 1 and 0 <= y and y <= self.paramDictionary["WorldSize"] - 1:
32                     tileVec.matrixVals[n][0] = copy(worldMap.tileArray[x][y])
33                     if [x,y] in enemyLocList:
34                         tileVec.matrixVals[n][0].WriteEnemy() # Writes enemies to tile if they exist
35                     else:
36                         tileVec.matrixVals[n][0] = blankOceanTile # Write water tile when out of range of the world -
37             n += 1
38         return tileVec
39
40     def TileVectorPostProcess(self, tileVec): # Returns 2 Vectors, 1 of tile types, 1 of grayscale values
41         tileTypeVec = Matrix(tileVec.order)
42         tileGrayscaleVec = Matrix(tileVec.order)
43
44         for n in range(tileVec.order[0]): # Converts vector to grayscale and type vectors
45             tileTypeVec.matrixVals[n][0] = tileVec.matrixVals[n][0].tileType
46
47             if tileVec.matrixVals[n][0].hasEnemy: # Enemy will overwrite tile colour if they are within that tile
48                 tileGrayscaleVec.matrixVals[n][0] = self.ColourToGrayscale(self.paramDictionary["ColourEnemy"])
49             else:
50                 tileGrayscaleVec.matrixVals[n][0] = self.ColourToGrayscale(tileVec.matrixVals[n][0].tileColour)
51
52         return tileTypeVec, tileGrayscaleVec
53

```

```

54     def ColourToGrayscale(self, colourTuple): # Converts colour value (255,255,255) to grayscale (0-1)
55         grayscale = (0.299 * colourTuple[0] + 0.587 * colourTuple[1] + 0.114 * colourTuple[2]) / 255
56         return grayscale
57
58     # Action Methods
59     def CommitAction(self, action, tileObjVec, worldMap, enemyList): # Commits the given Action
60         offset = self.paramDictionary["DQLOffset"]
61         sideLength = 2 * offset + 1
62
63         if action == 0:
64             self.Move(action, worldMap) # Move Up
65
66         elif action == 1:
67             self.Move(action, worldMap) # Move Right
68
69         elif action == 2:
70             self.Move(action, worldMap) # Move Down
71
72         elif action == 3:
73             self.Move(action, worldMap) # Move Left
74
75         elif action == 4 and tileObjVec.matrixVals[(sideLength * offset) + offset][0].hasObject == True: # Pickup
76             self.PickupItem(worldMap)
77
78         elif action == 5: # Attack Surroundings
79             self.Attack(enemyList)
80
81         elif action == 6: # Noop/Null action
82             pass
83             #print("Noop")
84
85     def Move(self, direction, worldMap): # Moves agent in given Direction
86         if direction == 0: self.location = [self.location[0], self.location[1] - 1] # Move Up
87         elif direction == 1: self.location = [self.location[0] + 1, self.location[1]] # Move Right
88         elif direction == 2: self.location = [self.location[0], self.location[1] + 1] # Move Down
89         elif direction == 3: self.location = [self.location[0] - 1, self.location[1]] # Move Left
90
91         self.alive = self.CheckIfValidStandTile(self.location, worldMap)
92         if not self.alive: return
93
94         if worldMap.tileArray[self.location[0]][self.location[1]].explored == False: # Checks if tile is explored
95             worldMap.tileArray[self.location[0]][self.location[1]].explored = True
96
97     def CheckIfValidStandTile(self, location, worldMap): # Checks if tile will murder the agent
98         x = location[0]
99         y = location[1]
100         if 0 <= x and x <= self.paramDictionary["WorldSize"] - 1 and 0 <= y and y <= self.paramDictionary["WorldS
101     else:
102         return False
103
104         if worldMap.tileArray[x][y].tileType == 0: # Checks if tile is water
105             return False
106
107         return True
108

```

```

109     def PickupItem(self, worldMap): # Pickup Item in the same tile as Agent
110         if worldMap.tileArray[self.location[0]][self.location[1]].hasObject:
111             self.inventory[worldMap.tileArray[self.location[0]][self.location[1]].objectType] += 1
112
113             worldMap.tileArray[self.location[0]][self.location[1]].ClearObject()
114
115     def Attack(self, enemyList): # Attacks in a given Area surrounding Agent
116         enemyLocList = [enemyList[i].location for i in range(len(enemyList))]
117
118         for y in range(self.location[1] - 1, self.location[1] + 2): # Loop through Tiles in surrounding area
119             for x in range(self.location[0] - 1, self.location[0] + 2):
120                 if [x,y] in enemyLocList:
121                     for i in range(len(enemyLocList)):
122                         if enemyLocList[i] == [x,y]:
123                             enemyList[i] = None
124
125         enemyList = [x for x in enemyList if x is not None] # Clears enemy list of None type
126
127     # Reward Method
128     def GetReward(self, action, tileObjVec): # Gets reward given action and tile vector
129         offset = self.paramDictionary["DQLOffset"]
130         sideLength = 2 * offset + 1
131
132         cumReward = 0
133
134         if action == 0: # Move Up
135             tile = tileObjVec.matrixVals[(sideLength * (offset - 1)) + offset][0]
136             cumReward += self.MoveReward(tile)
137
138         elif action == 1: # Move Right
139             tile = tileObjVec.matrixVals[(sideLength * offset) + offset + 1][0]
140             cumReward += self.MoveReward(tile)
141
142         elif action == 2: # Move Down
143             tile = tileObjVec.matrixVals[(sideLength * (offset + 1)) + offset][0]
144             cumReward += self.MoveReward(tile)
145
146         elif action == 3: # Move Left
147             tile = tileObjVec.matrixVals[(sideLength * offset) + offset - 1][0]
148             cumReward += self.MoveReward(tile)
149
150         elif action == 4: # Pickup Item
151             if tileObjVec.matrixVals[(sideLength * offset) + offset][0].hasObject:
152                 cumReward += self.paramDictionary["CollectItemReward"]
153             else:
154                 cumReward += self.paramDictionary["NoopReward"]
155
156         elif action == 5: # Attack
157             cumReward += self.CombatReward(tileObjVec)
158
159         elif action == 6: # No action/Noop/Idle
160             cumReward += self.paramDictionary["NoopReward"]
161
162         return cumReward
163

```

```

164     def MoveReward(self, tileObj): # Gets Reward given Agent moving into a tile
165         reward = 0
166         if tileObj.tileType == 0 or tileObj.hasEnemy: # Adds death reward if enemy or water
167             reward += self.paramDictionary["DeathReward"]
168         else: # Else adds explore and move reward
169             if tileObj.explored == False:
170                 reward += self.paramDictionary["ExploreReward"]
171             reward += self.paramDictionary["MoveReward"]
172         return reward
173
174     def CombatReward(self, tileObjVec):
175         killReward = self.paramDictionary["AttackReward"]
176         offset = self.paramDictionary["DQLOffset"]
177         sideLength = 2 * offset + 1
178
179         reward = 0
180
181         # Checks tiles around agent for enemies, adding reward where neccesary
182         if tileObjVec.matrixVals[(sideLength * (offset - 1)) + offset - 1][0].hasEnemy: reward += killReward
183         if tileObjVec.matrixVals[(sideLength * (offset - 1)) + offset][0].hasEnemy: reward += killReward
184         if tileObjVec.matrixVals[(sideLength * (offset - 1)) + offset + 1][0].hasEnemy: reward += killReward
185
186         if tileObjVec.matrixVals[(sideLength * offset) + offset - 1][0].hasEnemy: reward += killReward
187         if tileObjVec.matrixVals[(sideLength * offset) + offset][0].hasEnemy: reward += killReward
188         if tileObjVec.matrixVals[(sideLength * offset) + offset + 1][0].hasEnemy: reward += killReward
189
190         if tileObjVec.matrixVals[(sideLength * (offset + 1)) + offset - 1][0].hasEnemy: reward += killReward
191         if tileObjVec.matrixVals[(sideLength * (offset + 1)) + offset][0].hasEnemy: reward += killReward
192         if tileObjVec.matrixVals[(sideLength * (offset + 1)) + offset + 1][0].hasEnemy: reward += killReward
193
194         if reward > 0: return reward
195         else: return self.paramDictionary["AttackFailedReward"]
196
197     def GetRewardVector(self, tileObjVec, outputs): # Returns Vector of Reward Values Per action
198         returnVec = Matrix((outputs, 1))
199
200         for i in range(outputs):
201             returnVec.matrixVals[i][0] = self.GetReward(i, tileObjVec)
202
203         return returnVec
204
205     def MaxQ(self, rewardVec): # Used to get Max Reward from reward Vector
206         return max([rewardVec.matrixVals[i][0] for i in range(rewardVec.order[0])]) # Utilises List Comprehension
207
208     # Miscellaneous Methods
209     def Reset(self, worldMap): # Resets Inventory and Location of Agent
210         self.inventory = self.emptyInventory
211
212         self.location = Agent.SpawnPosition(worldMap)
213
214         self.alive = True
215
216     @staticmethod
217     def SpawnPosition(worldMap): # Returns a coord in which the Agent can spawn
218         spawnList = []

```

```

219
220     for y in range(0, worldMap.MAP_SIZE):
221         for x in range(0, worldMap.MAP_SIZE):
222             if worldMap.tileArray[x][y].tileType == 2:
223                 spawnList.append([x, y])
224
225     shuffle(spawnList)
226     return spawnList[0]

```

5.4 enemy.py

```

1  from newAgent import *
2  from random import randint
3
4  class Enemy(Agent): # Enemy inherits from Agent Class
5      def __init__(self, location, params): # Constructor for Enemy Class
6          self.paramDictionary = params
7
8          self.location = location
9
10         self.alive = True
11
12     def CommitAction(self, agent, worldMap): # Override of Agent Class method
13         xDif = agent.location[0] - self.location[0]
14         yDif = agent.location[1] - self.location[1]
15
16         if xDif == 0 and yDif == 0: # Checks if on Agent - If so -> Kill Agent
17             agent.alive = False
18             return
19
20         # Basic Path Finding for enemy
21         # Calculates difference between agent and player position, and moves in the greatest direction
22         if abs(xDif) > abs(yDif): # X Dif > Y Dif
23             if xDif > 0:
24                 self.location[0] += 1
25             else:
26                 self.location[0] -= 1
27         elif abs(xDif) < abs(yDif): # Y Dif > X Dif
28             if yDif > 0:
29                 self.location[1] += 1
30             else:
31                 self.location[1] -= 1
32         else: # Move random direction when X Dif = Y Dif
33             if randint(0,1):
34                 if xDif > 0:
35                     self.location[0] += 1
36                 else:
37                     self.location[0] -= 1
38             else:
39                 if yDif > 0:
40                     self.location[1] += 1
41                 else:
42                     self.location[1] -= 1
43

```

```

44         self.alive = self.CheckIfValidStandTile(self.location, worldMap) # Checks if walked into water or not
45
46     @staticmethod
47     def SpawnPosition(worldMap, enemyList): # Generate spawn position for the enemy given worldMap and enemyList
48         spawnList = []
49         enemyLocList = [enemyList[i].location for i in range(len(enemyList))]
50
51         for y in range(0, worldMap.MAP_SIZE):
52             for x in range(0, worldMap.MAP_SIZE):
53                 if worldMap.tileArray[x][y].tileType == 2: # Checks if tile type is
54                     spawnList.append([x, y])
55
56         shuffle(spawnList)
57
58         if spawnList[0] in enemyLocList: # Select spawn if not already selected
59             return None
60         else:
61             return spawnList[0]

```

5.5 worldClass.py

```

1  import json, random, pygame, threading
2  import perlinNoise
3
4  # Class to store Individual Tile Data
5  class Tile():
6      def __init__(self): # Initialise Tile object
7          self.tileHeight = -1
8          self.tileType = 0
9          self.tileColour = (0,0,0)
10         self.explored = False
11         self.hasObject = False
12         self.hasEnemy = False
13
14         def InitValues(self, tileType, height, colour): # Set/Initialise Tile Vales
15             self.tileType = tileType
16             self.tileHeight = height
17             self.tileColour = colour
18
19         def AddObject(self, objectType, objectColour): # Adds an Object to the Tile Object
20             self.hasObject = True
21             self.objectType = objectType
22             self.objectColour = objectColour
23
24         def ClearObject(self): # Clears Object from the Tile Object
25             self.hasObject = False
26             self.objectType = ""
27             self.objectColour = (0,0,0)
28
29         def WriteEnemy(self): # Write Enemy to tile
30             self.hasEnemy = True
31
32     # Class to store world terrain and object data
33     class WorldMap():

```

```

34     def __init__(self, seed, params): # Initialise method for creating an instance of the world
35         self.MAP_SIZE = params["WorldSize"]
36         self.TILE_WIDTH = params["TileWidth"]
37         self.MAP_SEED = seed
38         self.TILE_BORDER = params["TileBorder"]
39
40         self.tileArray = [[Tile() for i in range(self.MAP_SIZE)] for j in range(self.MAP_SIZE)]
41
42         self.paramDictionary = params
43
44     # Non Threaded Terrain Generation
45     def GenerateMap(self): # Generates terrain - Not Threaded
46         for y in range(0, self.MAP_SIZE):
47             for x in range(0, self.MAP_SIZE):
48                 xCoord = x / self.MAP_SIZE * self.paramDictionary["WorldScale"]
49                 yCoord = y / self.MAP_SIZE * self.paramDictionary["WorldScale"]
50
51                 self.tileArray[x][y].tileHeight = perlinNoise.OctaveNoise(self.MAP_SEED + xCoord, self.MAP_SEED +
52                                                                 self.paramDictionary["OctavesTerrain"], self.paramDic
53
54     # Threaded Terrain Generation
55     def GenerateThreadedParent(self): # Generates terrain using 4 threads
56         threads = []
57
58         halfMap = int(self.MAP_SIZE / 2)
59         fullMap = self.MAP_SIZE
60
61         # Create 4 threads for threaded child functions
62         threads.append(threading.Thread(target=self.ThreadedChild, args=(0, halfMap, 0, halfMap)))
63         threads.append(threading.Thread(target=self.ThreadedChild, args=(halfMap, fullMap, 0, halfMap)))
64         threads.append(threading.Thread(target=self.ThreadedChild, args=(0, halfMap, halfMap, fullMap)))
65         threads.append(threading.Thread(target=self.ThreadedChild, args=(halfMap, fullMap, halfMap, fullMap)))
66
67
68         # Start all the threads
69         for t in threads:
70             t.start()
71
72         # While threads arent finished, pause
73         while threading.activeCount() > 1:
74             pass
75
76         self.RenderMap() # Render Map
77
78     def ThreadedChild(self, x1, x2, y1, y2): # Child Method to GenerateThreadedParent
79         for y in range(y1, y2):
80             for x in range(x1, x2):
81                 xCoord = (x / self.MAP_SIZE) * self.paramDictionary["WorldScale"]
82                 yCoord = (y / self.MAP_SIZE) * self.paramDictionary["WorldScale"]
83
84                 self.tileArray[x][y].tileHeight = perlinNoise.OctaveNoise(self.MAP_SEED + xCoord + self.time, self
85                                                                 self.paramDictionary["OctavesTerrain"], self.paramDic
86
87     # Generate Tree Methods
88     def GenerateTreeArea(self): # Uses perlin noise to generate the areas for trees to spawn in

```

```

89     TSO = self.paramDictionary["TreeSeedOffset"]
90
91     treeList = []
92
93     for y in range(0, self.MAP_SIZE):
94         for x in range(0, self.MAP_SIZE):
95             xCoord = x / self.MAP_SIZE
96             yCoord = y / self.MAP_SIZE
97
98             temp = perlinNoise.OctaveNoise(self.MAP_SEED + xCoord + TSO, self.MAP_SEED + yCoord + TSO,
99                                             self.paramDictionary["OctavesTrees"], self.paramDictionary["PersistenceTrees"]) # Sam
100
101             tileValue = self.Clamp(((self.tileArray[x][y].tileHeight / 2) + 0.5), 0.0, 1.0) # Clamp value
102
103             if (temp > self.paramDictionary["TreeHeight"] and tileValue > self.paramDictionary["Coast"] + self
104                                     tileValue < self.paramDictionary["Grass"] - s
105
106                 treeList.append([x, y])
107
108     poissonArray = self.PoissonDiscSampling(treeList) # Get Poisson Disc Sampling values for poisson array
109
110     for y in range(0, self.MAP_SIZE):
111         for x in range(0, self.MAP_SIZE):
112             self.tileArray[x][y].ClearObject() # Clear Existing objects from tile map
113
114             if poissonArray[x][y] == True:
115                 self.tileArray[x][y].AddObject(self.paramDictionary["TreeType"], self.paramDictionary["Colour"]
116
117 def PoissonDiscSampling(self, pointList): # A tweaked version of poisson disc sampling in 2 dimensions
118     k = self.paramDictionary["PoissonKVal"]
119
120     pickedPoints = [[False for i in range(self.MAP_SIZE)] for j in range(self.MAP_SIZE)] # Blank array of Fal
121
122     numPoints = len(pointList) - 1
123     if numPoints <= 0: # Catches if no points
124         return pickedPoints
125
126     sampleNum = 0
127
128     while sampleNum <= k: # While sampled attempts is less than k
129         sample = pointList[random.randint(0, numPoints)]
130
131         result = self.PoissonCheckPoint(sample, pickedPoints) # Check points
132         if result == True:
133             pickedPoints[sample[0]][sample[1]] = True
134             sampleNum = 0
135             continue
136         else:
137             sampleNum += 1
138             continue
139
140     return pickedPoints
141
142 def PoissonCheckPoint(self, point, pickedPoints): # Checks Specific points around a point for objects
143     if (1 <= point[0] and point[0] <= self.paramDictionary["WorldSize"] - 2 and
144         1 <= point[1] and point[1] <= self.paramDictionary["WorldSize"] - 2):

```



```

144         if pickedPoints[point[0]][point[1] - 1] == True: return False
145         elif pickedPoints[point[0] + 1][point[1]] == True: return False
146         elif pickedPoints[point[0]][point[1] + 1] == True: return False
147         elif pickedPoints[point[0] - 1][point[1]] == True: return False
148         elif pickedPoints[point[0]][point[1]] == True: return False
149         else: return True
150
151     # Render Methods
152     def RenderMap(self): # Renders terrain onto Pygame surface
153         resolution = self.MAP_SIZE * self.TILE_WIDTH
154         self.RenderedMap = pygame.Surface((resolution, resolution))
155         self.RenderedMap.set_colorkey((0,0,0))
156
157         if self.paramDictionary["Grayscale"] == 1: # Renders in grayscale if specified
158             for y in range(0, self.MAP_SIZE):
159                 for x in range(0, self.MAP_SIZE):
160                     value = self.tileArray[x][y].tileHeight
161                     value = (value / 2) + 0.5
162                     value = self.Clamp(value, 0.0, 1.0)
163
164                     pygame.draw.rect(self.RenderedMap, (255 * value, 255 * value, 255 * value), ((x * self.TILE_W
165                     (y * self.TILE_WIDTH + self.TILE_BORDER), self.TILE_WIDTH - (self.TILE_BORDER * 2), s
166
167         else: # Else renders in Colour
168             for y in range(0, self.MAP_SIZE):
169                 for x in range(0, self.MAP_SIZE):
170                     value = self.tileArray[x][y].tileHeight
171                     value = (value / 2) + 0.5
172                     value = self.Clamp(value, 0.0, 1.0) # Clamps value between 0 and 1
173
174                     colour = None
175
176                     if value == 0: # Colour ramp for all available colours
177                         colour = (0,0,0)
178                     elif value < self.paramDictionary["Water"]:
179                         colour = tuple(self.paramDictionary["ColourWater"])
180                         self.tileArray[x][y].tileType = 0
181                         self.tileArray[x][y].tileColour = colour
182                     elif value < self.paramDictionary["Coast"]:
183                         colour = tuple(self.paramDictionary["ColourCoast"])
184                         self.tileArray[x][y].tileType = 1
185                         self.tileArray[x][y].tileColour = colour
186                     elif value < self.paramDictionary["Grass"]:
187                         colour = tuple(self.paramDictionary["ColourGrass"])
188                         self.tileArray[x][y].tileType = 2
189                         self.tileArray[x][y].tileColour = colour
190                     elif value < self.paramDictionary["Mountain"]:
191                         colour = tuple(self.paramDictionary["ColourMountain"])
192                         self.tileArray[x][y].tileType = 3
193                         self.tileArray[x][y].tileColour = colour
194
195                     # Draws correct colour pixel to rendered map - takes into account width and border
196                     pygame.draw.rect(self.RenderedMap, colour, ((x * self.TILE_WIDTH + self.TILE_BORDER),
197                     (y * self.TILE_WIDTH + self.TILE_BORDER), self.TILE_WIDTH - (self.TILE_BORDER * 2), s
198

```

```

199     def RenderInteractables(self): # Renders interactables onto pygame surface
200         resolution = self.MAP_SIZE * self.TILE_WIDTH
201         self.RenderedInteractables = pygame.Surface((resolution, resolution))
202         self.RenderedInteractables.set_colorkey((0,0,0))
203
204         ITB = self.paramDictionary["InteractableTileBorder"]
205
206         for y in range(0, self.MAP_SIZE): # Draw interactables to rendered image
207             for x in range(0, self.MAP_SIZE):
208                 if self.tileArray[x][y].hasObject == True:
209                     tile = self.tileArray[x][y]
210                     pygame.draw.rect(self.RenderedInteractables, tile.objectColour, ((x * self.TILE_WIDTH + ITB),
211                                             (y * self.TILE_WIDTH + ITB), self.TILE_WIDTH - (ITB * 2), self.TILE_WIDTH - (ITB * 2))
212
213     def DrawMap(self, window): # Blits the rendered frames onto the passed through window
214         window.blit(self.RenderedMap, (0,0))
215         self.RenderInteractables()
216         window.blit(self.RenderedInteractables, (0,0))
217
218     # Miscellaneous Methods
219     def Clamp(self, val, low, high): # Simple function to clamp a value between two numbers - Used to make sure n
220         return low if val < low else high if val > high else val

```

5.6 perlinNoise.py

```

1  import random, math
2
3  p = [151,160,137,91,90,15,
4      131,13,201,95,96,53,194,233,7,225,140,36,103,30,69,142,8,99,37,240,21,10,23,
5      190, 6,148,247,120,234,75,0,26,197,62,94,252,219,203,117,35,11,32,57,177,33,
6      88,237,149,56,87,174,20,125,136,171,168, 68,175,74,165,71,134,139,48,27,166,
7      77,146,158,231,83,111,229,122,60,211,133,230,220,105,92,41,55,46,245,40,244,
8      102,143,54, 65,25,63,161, 1,216,80,73,209,76,132,187,208, 89,18,169,200,196,
9      135,130,116,188,159,86,164,100,109,198,173,186, 3,64,52,217,226,250,124,123,
10     5,202,38,147,118,126,255,82,85,212,207,206,59,227,47,16,58,17,182,189,28,42,
11     223,183,170,213,119,248,152, 2,44,154,163, 70,221,153,101,155,167, 43,172,9,
12     129,22,39,253, 19,98,108,110,79,113,224,232,178,185, 112,104,218,246,97,228,
13     251,34,242,193,238,210,144,12,191,179,162,241, 81,51,145,235,249,14,239,107,
14     49,192,214, 31,181,199,106,157,184, 84,204,176,115,121,50,45,127, 4,150,254,
15     138,236,205,93,222,114,67,29,24,72,243,141,128,195,78,66,215,61,156,180]
16  p = p + p
17
18  def OctaveNoise(x, y, octaves, persistence): # Sums multiple levels of perlin noise
19      total = 0
20      frequency = 1
21      amplitude = 1
22      maxValue = 0
23
24      for i in range(octaves): # Combines Multiple octaves of perlin noise
25          total += ((Noise(x * frequency, y * frequency)) * amplitude)
26
27          maxValue += amplitude
28
29          amplitude *= persistence

```

```

30         frequency *= 2
31
32     return total / maxValue
33
34 def Noise(x, y): # Returns a value of the perlin noise function at (x, y) coordinate
35     xi = math.floor(x) % 255
36     yi = math.floor(y) % 255
37
38     g1 = p[p[xi] + yi]
39     g2 = p[p[xi + 1] + yi]
40     g3 = p[p[xi] + yi + 1]
41     g4 = p[p[xi + 1] + yi + 1]
42
43     xf = x - math.floor(x)
44     yf = y - math.floor(y)
45
46     d1 = Grad(g1, xf, yf)
47     d2 = Grad(g2, xf - 1, yf)
48     d3 = Grad(g3, xf, yf - 1)
49     d4 = Grad(g4, xf - 1, yf - 1)
50
51     u = Fade(xf)
52     v = Fade(yf)
53
54     x1Inter = Lerp(u, d1, d2)
55     x2Inter = Lerp(u, d3, d4)
56     yInter = Lerp(v, x1Inter, x2Inter)
57
58     return yInter
59
60 def Grad(hash, x, y): # Gradient Function defined as part of the algorithm
61     temp = hash & 3
62     if temp == 0:
63         return x + y
64     elif temp == 1:
65         return -x + y
66     elif temp == 2:
67         return x - y
68     elif temp == 3:
69         return -x - y
70     else:
71         return 0
72
73 def Lerp(ammount, left, right): # Linear interpolation of values
74     return ((1 - ammount) * left + ammount * right)
75
76 def Fade(t): # Fade Function defined as part of the algorithm
77     return t * t * t * (t * (t * 6 - 15) + 10)

```

5.7 deepqlearning.py

```

1  from audioop import bias
2  import random, pickle, math
3  from typing import final

```

```

4  from matrix import Matrix
5  import activations
6  from copy import copy
7  from datalogger import *
8  import time
9
10 class DoubleNeuralNet(): # Wraps a Main and Target Neural Network together
11     def __init__(self, layers, params, load=False, loadName="DQNetwork"): # Constructor for a Double Neural Netwo
12         self.paramDictionary = params
13
14         if not load: # Create brand new values
15             self.MainNetwork = NeuralNet(layers, params)
16             self.TargetNetwork = NeuralNet(layers, params)
17
18             self.ExperienceReplay = Deque(self.paramDictionary["ERBuffer"])
19
20             self.epsilon = self.paramDictionary["DQLEpsilon"]
21
22             self.step = 0
23             self.cumReward = 0.0
24
25             self.layerActivation = activations.Sigmoid()
26             self.finalLayerActivation = activations.SoftMax()
27         else:
28             self.LoadState(loadName) # Load values from saved data
29
30         self.fileName = loadName
31
32         self.activations = (self.layerActivation, self.finalLayerActivation) # Tuple of activations
33
34         self.batchReward = 0
35         self.maxBatchReward = 0
36         self.batchLoss = 0
37         self.dataPoints = []
38
39                                     # BatchReward, MaxBatchReward, PercentageDifference, Step
40         self.actionTracker = DataLogger("ActionTracker", [[float, int], [float, int], [float, int], int], False)
41
42         self.startTime = time.time()
43
44     def TakeStep(self, agent, worldMap, enemyList): # Takes a step forward in time
45         self.step += 1
46
47         # Forward Propagation
48         agentSurround = agent.GetTileVector(worldMap, enemyList)
49         postProcessedSurround = agent.TileVectorPostProcess(agentSurround) # Retrieve Vector of State info from A
50         netInput = postProcessedSurround[1]
51
52         self.MainNetwork.ForwardPropagation(netInput, self.activations) # Forward Prop the Main Network
53
54         output = self.MainNetwork.layers[-1].activations
55         #print(output)
56         outputMax = output.MaxInVector()
57
58         # Action Taking and Reward

```

```

59     if random.random() > self.epsilon:
60         softmaxed = self.finalLayerActivation.Activation(copy(output))
61         action = random.randint(0, self.paramDictionary["DeepQLearningLayers"][-1] - 1)
62         val = random.random()
63         totalled = 0
64         for i in range(softmaxed.order[0]):
65             totalled += softmaxed.matrixVals[i][0]
66             if totalled >= val:
67                 action = i
68                 break
69     else:
70         action = random.randint(0, self.paramDictionary["DeepQLearningLayers"][-1] - 1)
71
72     rewardVector = agent.GetRewardVector(agentSurround, self.paramDictionary["DeepQLearningLayers"][-1])
73     reward = rewardVector.matrixVals[action][0] # Get reward given action
74     self.cumReward += reward
75     self.batchReward += reward
76     self.maxBatchReward += rewardVector.MaxInVector()[0]
77
78     agent.CommitAction(action, agentSurround, worldMap, enemyList) # Take Action
79     # Epsilon Regression
80     self.epsilon *= self.paramDictionary["DQLEpisonRegression"]
81
82     # Assigning values to tempExperience
83     tempExp = Experience()
84     tempExp.state = agentSurround
85     tempExp.action = action
86     tempExp.reward = rewardVector
87     tempExp.stateNew = agent.GetTileVector(worldMap, enemyList)
88
89     self.ExperienceReplay.PushFront(copy(tempExp))
90
91     # Back Propagation
92     expectedValues = self.ExpectedValue(output, tempExp, agent) # Calculating Loss
93
94     Cost = self.HalfSquareDiff(output, expectedValues)
95
96     self.batchLoss += Cost.Sum()
97
98     self.MainNetwork.layers[-1].errSignal = Cost * self.layerActivation.Derivative(copy(self.MainNetwork.layers[-1].activation))
99
100    self.MainNetwork.BackPropagationV2(self.activations) # Back Propagating the loss
101
102    # Do things every X steps passed
103    if self.step % self.paramDictionary["TargetReplaceRate"] == 0: # Replace Weights in Target Network
104        self.TargetNetwork.layers = self.MainNetwork.layers
105
106    # Sample Experience Replay Buffer
107    if (self.paramDictionary["EREnabled"] and self.step % self.paramDictionary["ERSampleRate"] == 0 and self.batchLoss > 0):
108        self.SampleExperienceReplay(agent)
109
110    # Actions to run after every Batch
111    if self.step % self.paramDictionary["DQLEpoch"] == 0:
112        print(self.step, self.cumReward, self.epsilon, time.time() - self.startTime)
113

```

```

114         self.MainNetwork.UpdateWeightsAndBiases(self.paramDictionary["DQLEpoch"]) # Update weights and biases
115
116         if self.paramDictionary["SaveWeights"]: # Saves weights if specified
117             self.SaveState(self.fileName)
118
119         #Log Action
120         self.actionTracker.LogDataPoint([self.batchReward, self.maxBatchReward, self.batchLoss, self.step])
121         #self.actionTracker.LogDataPointBatch(self.dataPoints)
122
123         self.dataPoints = []
124         self.actionTracker.SaveDataPoints()
125
126         self.batchReward = 0
127         self.maxBatchReward = 0
128         self.batchLoss = 0
129
130     def SampleExperienceReplay(self, agent): # Samples the Experience Replay Buffer, Back Propagating its Finding
131         samples = self.ExperienceReplay.Sample(self.paramDictionary["ERSampleSize"])
132
133         for sample in samples:
134             postProcessedSurround = agent.TileVectorPostProcess(sample.state) # Post process the Tile Vector
135             netInput = postProcessedSurround[1]
136
137             self.MainNetwork.ForwardPropagation(netInput, self.activations) # Forward Prop the Main Network
138
139             output = self.MainNetwork.layers[-1].activations
140
141             expectedValues = self.ExpectedValue(output, sample, agent) # Calculating Loss
142
143             Cost = self.HalfSquareDiff(output, expectedValues)
144
145             self.MainNetwork.layers[-1].errSignal = Cost * self.layerActivation.Derivative(copy(self.MainNetwork.
146
147             self.MainNetwork.BackPropagationV2(self.activations) # Back Propagating the loss
148
149     def HalfSquareDiff(self, networkOutput, expected):
150         return ((expected - networkOutput) ** 2) * 0.5
151
152     def ExpectedValue(self, output, tempExp, agent):
153         #  $L^i(W^i) = ((r + \gamma \max_{a'} Q(s', a'; W^{i-1}) - Q(s, a, W)) ** 2$ 
154         #  $Loss = ((Reward[] + Gamma * MaxQ(s', a'; TNet)) - Q(s, a)) ^ 2$ 
155
156         Reward = tempExp.reward
157         Gamma = self.paramDictionary["DQLGamma"]
158
159         #self.TargetNetwork.ForwardPropagation(agent.TileVectorPostProcess(tempExp.state)[1], self.activations) #
160
161         #targetNetAction = self.TargetNetwork.layers[-1].activations.MaxInVector()[1]
162
163
164         tempRewardVec = agent.GetRewardVector(tempExp.stateNew, self.paramDictionary["DeepQLearningLayers"][-1])
165         maxQTNet = agent.MaxQ(tempRewardVec) # Max of Target network
166
167         LossVec = ((Reward + (Gamma * maxQTNet)) - output) ** 2 # Bellman Equation
168         return LossVec

```

```

169
170     def SaveState(self, file):
171         state = [self.MainNetwork, self.TargetNetwork, self.ExperienceReplay, self.step,
172                 self.epsilon, self.cumReward, self.layerActivation, self.finalLayerActivation]
173         with open("DQLearningData\\" + file + ".dqn", "wb") as f:
174             pickle.dump(state, f)
175
176     def LoadState(self, file): # Returns stored Neural Network data
177         with open("DQLearningData\\" + file + ".dqn", "rb") as f:
178             state = pickle.load(f)
179
180             self.MainNetwork = state[0]
181             self.TargetNetwork = state[1]
182             self.ExperienceReplay = state[2]
183             self.step = state[3]
184             self.epsilon = state[4]
185             self.cumReward = state[5]
186             self.layerActivation = state[6]
187             self.finalLayerActivation = state[7]
188
189     class NeuralNet(): # Neural Network Implementation
190         def __init__(self, layersIn, params): # Constructor for a Single Neural Network
191             self.paramDictionary = params
192
193             newLayersIn = copy(layersIn)
194
195             newLayersIn.append(1)
196
197             self.layers = []
198
199             for i in range(len(newLayersIn) - 1):
200                 print(newLayersIn[i])
201                 self.layers.append(Layer(newLayersIn[i], newLayersIn[i + 1]))
202
203         def ForwardPropagation(self, inputVector, activations): # Iterates through Forward Propagation
204             self.layers[0].activations = inputVector
205
206             for i in range(0, len(self.layers) - 1):
207                 self.layers[i].ForwardPropagation(self.layers[i+1], activations)
208
209             #self.layers[-1].ForwardPropagation(self.layers[-2], activations, finalLayer=True)
210
211         def BackPropagationV2(self, activations): # Iterates through Back Propagation V2
212             self.layers[-2].BackPropagationV2(self.layers[-1], self.paramDictionary["DQLLearningRate"], activations)
213
214             for i in range(len(self.layers) - 3, 0, -1):
215                 self.layers[i].BackPropagationV2(self.layers[i+1], self.paramDictionary["DQLLearningRate"], activations)
216
217         def UpdateWeightsAndBiases(self, epochCount): # Update Weights and biases
218             for i in range(1, len(self.layers)):
219                 self.layers[i].UpdateWeightsAndBiases(epochCount)
220
221     class Layer(): # Layer for a Neural Network
222         def __init__(self, size, nextSize, inputLayer=False): # Constructor for a Layer Object
223             if inputLayer == False: # Additional objects if not the input layer

```

```

224         pass
225
226     self.weightMatrix = Matrix((nextSize, size), random=True)
227     self.biasVector = Matrix((nextSize, 1), random=False)
228
229     self.weightUpdates = Matrix((nextSize, size))
230     self.biasUpdates = Matrix((nextSize, 1))
231
232     self.errSignal = Matrix((nextSize, 1))
233     self.preactivations = Matrix((size, 1))
234     self.activations = Matrix((size, 1))
235
236     def ForwardPropagation(self, nextLayer, activations): # Forward Propagates the Neural Network
237         self.preactivations = self.weightMatrix * self.activations + self.biasVector
238
239         nextLayer.activations = activations[0].Activation(copy(self.preactivations))
240
241     def BackPropagationV2(self, prevLayer, lr, layerActivations): # 2nd Revision of Back Propagation
242         deltaWeightProduct = (prevLayer.weightMatrix.Transpose() * prevLayer.errSignal)
243         self.errSignal = deltaWeightProduct * layerActivations[0].Derivative(copy(self.preactivations))
244
245         weightDerivatives = self.errSignal * self.activations.Transpose()
246         biasDerivatives = self.errSignal
247
248         self.weightUpdates += weightDerivatives * lr
249         self.biasUpdates += biasDerivatives * lr
250
251     def UpdateWeightsAndBiases(self, epochCount): # Update Weights and Biases
252         self.weightMatrix -= (self.weightUpdates * (1 / epochCount))
253         self.biasVector -= (self.biasUpdates * (1 / epochCount))
254
255         self.weightUpdates.Clear()
256         self.biasUpdates.Clear()
257
258     class Experience(): # Used in Experience Replay
259         def __init__(self, state = None, action = None, reward = None, stateNew = None): # Constructor for an Experience
260             self.state = state
261             self.action = action
262             self.reward = reward
263             self.stateNew = stateNew
264
265     class Deque(): # Partial Double Ended Queue Implementation
266         def __init__(self, length):
267             self.length = length
268
269             self.queue = [None for i in range(self.length)]
270
271             self.frontP = -1
272             self.backP = -1
273
274         def PushFront(self, item): # Pushes item to front of Queue
275             self.frontP = (self.frontP + 1) % self.length
276
277             if self.queue[self.frontP] != None:
278                 self.backP = (self.frontP + 1) % self.length

```



```

279
280     self.queue[self.frontP] = item
281
282     def Full(self): # Checks if Queue is full
283         if self.queue[self.length - 1] != None:
284             return True
285         return False
286
287     def First(self): # Returns Front Item from Queue
288         return self.queue[self.frontP]
289
290     def Last(self): # Returns Final Item from Queue
291         return self.queue[(self.frontP + 1) % self.length]
292
293     def Sample(self, n): # Samples N number of samples from the deque
294         temp = self.queue
295         return random.sample(temp, n)

```

5.8 activations.py

```

1  from abc import ABC, abstractmethod
2  from math import e, tanh, exp, cosh
3  from matrix import *
4
5  class Activation(ABC): # Abstract Base Class
6      @abstractmethod
7      def Activation(self, x): # Abstract Activation Method
8          pass
9
10     @abstractmethod
11     def Derivative(self, x): # Abstract Derivative Method
12         pass
13
14     class ReLu(Activation): # ReLu
15         def __init__(self):
16             pass
17
18         def Activation(self, x): # Returns value if greater than 0, else 0
19             for row in range(x.order[0]):
20                 x.matrixVals[row][0] = max(0, x.matrixVals[row][0])
21             return x
22
23         def Derivative(self, x): # If value is greater than 0 return 1, else return 0
24             for row in range(x.order[0]):
25                 if x.matrixVals[row][0] > 0: x.matrixVals[row][0] = 1
26                 else: 0
27             return x
28
29     class LeakyReLu(Activation): # Leaky ReLu
30         def __init__(self):
31             pass
32
33         def Activation(self, x): # Returns value if greater than 0, else a apply a gradient to x and return it
34             for row in range(x.order[0]):

```

```

35         x.matrixVals[row][0] = max(x.matrixVals[row][0] * 0.1, x.matrixVals[row][0])
36     return x
37
38     def Derivative(self, x): # If value is greater than 0 return 1, else return 0.01
39         for row in range(x.order[0]):
40             if x.matrixVals[row][0] > 0: x.matrixVals[row][0] = 1
41             else: 0.1
42         return x
43
44 class Sigmoid(Activation): # Sigmoid
45     def __init__(self):
46         pass
47
48     def Activation(self, x): # Mathematical Function to get "squish" values between 0 and 1
49         for row in range(x.order[0]):
50             if x.matrixVals[row][0] > 15: x.matrixVals[row][0] = 1
51             elif x.matrixVals[row][0] < -15: x.matrixVals[row][0] = 0
52             else: x.matrixVals[row][0] = 1 / (1 + exp(-x.matrixVals[row][0]))
53         return x
54
55     def Derivative(self, x): # Derivative of the Sigmoid Function
56         for row in range(x.order[0]):
57             sigmoidSingle = self.ActivationSingle(x.matrixVals[row][0])
58             x.matrixVals[row][0] = sigmoidSingle * (1 - sigmoidSingle)
59         return x
60
61     def ActivationSingle(self, x): # Single value for use in the derivative
62         if x > 15: return 1
63         elif x < -15: return 0
64         else: return 1 / (1 + exp(-x))
65
66 class SoftMax(Activation): # SoftMax
67     def __init__(self):
68         pass
69
70     def Activation(self, x): # Returns a probability distribution between a vector of values totalling to 1
71         sumToK = 0
72
73         for i in range(x.order[0]):
74             sumToK += exp(x.matrixVals[i][0])
75
76         outVector = Matrix(x.order)
77
78         for i in range(x.order[0]):
79             outVector.matrixVals[i][0] = (exp(x.matrixVals[i][0])) / sumToK
80
81         return outVector # Returns vector and best index
82
83     def Derivative(self, x): # Derivative of the softmax function
84         for row in range(x.order[0]):
85             x.matrixVals[row][0] = x.matrixVals[row][0] * (1 - x.matrixVals[row][0])
86
87         return x
88
89 class NullActivation(Activation): # No activation function

```

```

90     def __init__(self):
91         pass
92
93     def Activation(self, x): # Returns the same values
94         return x
95
96     def Derivative(self, x): # Returns the same values
97         return 1
98
99 class TanH(Activation): # TanH
100     def __init__(self):
101         pass
102
103     def Activation(self, x): # TanH mathematical function
104         for row in range(x.order[0]):
105             x.matrixVals[row][0] = tanh(x.matrixVals[row][0])
106         return x
107
108     def Derivative(self, x): # Derivative of TanH
109         for row in range(x.order[0]):
110             x.matrixVals[row][0] = (1 / (cosh(x.matrixVals[row][0]))) ** 2
111         return x

```

5.9 datalogger.py

```

1  import pickle, random
2  from heap import *
3
4  # Data Logger Class for logging information for analysis
5  class DataLogger():
6      def __init__(self, name, dataStructure, load=True): # Constructor Method
7          self.name = name
8
9          self.dataStructure = dataStructure
10
11          if load: # Loads Data if available but else create blank
12              self.dataPoints = DataLogger.LoadDataPoints(name)
13          else:
14              self.dataPoints = []
15
16      def LogDataPointBatch(self, dataPoints): # Logs a Batch of Data Points
17          for i in range(len(dataPoints)):
18              self.LogDataPoint(dataPoints[i])
19
20      def LogDataPoint(self, dataPoint): # Logs Data Point to Data Point list
21          if self.CheckMatchStructure(dataPoint):
22              self.dataPoints.append(dataPoint)
23
24
25      def CheckMatchStructure(self, dataPoint): # Checks the given Data Point is in the correct Form
26          if len(dataPoint) != len(self.dataStructure): # Throws error if lengths dont match
27              raise Exception("Structure of Data Point does not match Collector Specified Structure")
28
29          for i in range(len(dataPoint)):

```

```

30         t1 = type(dataPoint[i]) # Type 1
31         t2 = self.dataStructure[i] # Type 2
32
33         if t1 == list and type(t2) != list: # Checks if list is all of same type
34             flag = False
35
36             for x in range(len(dataPoint[i])):
37                 if type(dataPoint[i][x]) != t2:
38                     flag = True
39             if not flag:
40                 continue
41
42         elif t1 == list and type(t2) == list: # Checks list against list
43             if len(dataPoint[i]) == len(t2):
44                 flag = False
45                 for x in range(len(dataPoint[i])):
46                     if type(dataPoint[i][x]) != t2[x]:
47                         flag = True
48
49             if not flag:
50                 continue
51
52         elif type(t2) == list: # Checks Multiple types against t1
53             flag = False
54
55             for x in range(len(t2)):
56                 if t1 == t2[x]:
57                     flag = True
58             if flag:
59                 continue
60
61         else: # Checks Singular type against t1
62             if t1 == t2:
63                 continue
64
65         raise Exception(("Type: {} != Data Structure Type: {} \n {}".format(t1, t2, self.dataStructure)))
66     return True
67
68 def HeapSort(self, parameterIndex): # O(n*log n) sorting algorithm utilising a Heap Data structure, Sorts the
69     # 1000 Items -> 0.13
70     # 10000 Items -> 12.1
71     # 100000 Items -> 1646 or 27.4 minutes
72
73     if type(self.dataStructure[parameterIndex]) == list: # Throw error if data structure element is List
74         raise Exception("Cannot sort by structure: {}".format(type(self.dataStructure[parameterIndex])))
75
76     elif self.dataStructure[parameterIndex] == bool: # Throw error if data structure element is Bool
77         raise Exception("Cannot sort by structure: {}".format(self.dataStructure[parameterIndex]))
78
79     sortedList = []
80
81     heap = Heap(self.dataPoints, parameterIndex) # Creates a new heap
82
83     while heap.Length() - 1 >= 0:
84         sortedList.append(heap.RemoveTop()) # Loops popping and appending greatest element from Heap

```

```

85
86     return sortedList
87
88     def Select(self, searchIndex, searchContents): # Select a specified element with contents from data points
89         returnedList = []
90
91         for i in range(len(self.dataPoints)):
92             if self.dataPoints[i][searchIndex] in searchContents:
93                 returnedList.append(self.dataPoints[i])
94
95         return returnedList
96
97     # Using Pickle to Save/Load
98     @staticmethod
99     def LoadDataPoints(file): # Returns stored dataPoints
100         with open("DataLogger\\" + file + ".data", "rb") as f:
101             temp = pickle.load(f)
102             return temp
103
104     def SaveDataPoints(self): # Saves dataPoints to a file
105         with open("DataLogger\\" + self.name + ".data", "wb") as f:
106             pickle.dump(self.dataPoints, f)

```

5.10 heap.py

```

1  import math
2
3  # A Binary tree with the heap property, such that for every element, both children are <= to the parent
4  class Heap:
5      def __init__(self, elements, indexIn): # Creates a new heap from a list of elements, and assigns an index for
6          self.elements = elements
7          self.index = indexIn
8
9          self.Heapify()
10
11     def AddElement(self, element): # Adds Singular element to Heap
12         self.elements.append(element)
13         self.SiftUp(len(self.elements) - 1)
14
15     def SiftUp(self, elementIndex): # Sifts a singular element up the heap if possible
16         newElementIndex = elementIndex
17         isHeap = False
18
19         while not isHeap: # Repeat until is a heap again
20             parentIndex = math.floor((newElementIndex - 1) / 2)
21
22             if parentIndex == 0 and newElementIndex == 0: # Base Case
23                 isHeap = True
24
25             elif self.elements[newElementIndex][self.index] >= self.elements[parentIndex][self.index]: # Swaps el
26                 tempSwap = self.elements[parentIndex]
27                 self.elements[parentIndex] = self.elements[newElementIndex]
28                 self.elements[newElementIndex] = tempSwap
29

```

```

30         newElementIndex = parentIndex
31     else:
32         isHeap = True
33
34 def SiftDown(self, elementIndex): # Sifts a singular element down the heap if possible
35     rootIndex = elementIndex
36     isHeap = False
37
38     end = len(self.elements) - 1
39
40     while ((2 * rootIndex) + 1) <= end: # Repeat until the next root index is outside the dimensions of the h
41         childIndex = (rootIndex * 2) + 1
42
43         if childIndex + 1 <= end and self.elements[childIndex][self.index] < self.elements[childIndex + 1][se
44             childIndex += 1
45
46         if self.elements[rootIndex][self.index] < self.elements[childIndex][self.index]: # Swapping elements
47             tempSwap = self.elements[childIndex]
48             self.elements[childIndex] = self.elements[rootIndex]
49             self.elements[rootIndex] = tempSwap
50
51         rootIndex = childIndex
52     else:
53         break
54
55 def RemoveTop(self): # Pops top element off of Heap and returns it, heapifies the heap once removed
56     tempSwap = self.elements[-1]
57     self.elements[-1] = self.elements[0] # Swaps First and Last elements
58     self.elements[0] = tempSwap
59
60     returnElement = self.elements[-1] # Stores and deletes the final element
61     self.elements = self.elements[:-1]
62
63     self.Heapify() # Creates Heap again
64
65     return returnElement # Returns Top element
66
67 def Peek(self): # Returns root/top element
68     return self.elements[0]
69
70 def Length(self): # Returns size of heap
71     return len(self.elements)
72
73 def Heapify(self): # Returns values to a heap form, where all children of parents are less than or equal too
74     for i in range(math.floor((len(self.elements) - 1) / 2), -1, -1):
75         self.SiftDown(i)

```

5.11 plotData.py

```

1 import matplotlib.pyplot as plt
2 import pickle
3 from os import listdir
4 from os.path import isfile, join
5 from typing import DefaultDict

```

```
6
7 def LoadFileList(dir): # Locating files in directory and returning them as a dictionary
8     directoryList = listdir(dir)
9     validFiles = [f for f in directoryList if isfile(join(dir, f))]
10
11     fileDict = DefaultDict(str)
12
13     for i in range(len(validFiles)):
14         fileDict[i] = validFiles[i]
15
16     return fileDict
17
18 def PickChoice(fileDict): # Pick choice from file dictionary
19     print("List of Data Files:")
20     for file in fileDict:
21         print(str(file) + " : " + fileDict[file])
22
23     inp = eval(input())
24     if isinstance(inp, int):
25         return fileDict[inp]
26     else:
27         raise Exception("Not a valid input")
28
29 def LoadPoints(file): # Load Data Points from file
30     dataPoints = []
31     with open("DataLogger\\" + file, "rb") as f:
32         dataPoints = pickle.load(f)
33     return dataPoints
34
35 # Logic
36 fileDictionary = LoadFileList("DataLogger\\")
37 file = PickChoice(fileDictionary)
38 dataPoints = LoadPoints(file)
39
40 print("Plot: ")
41 inp = eval(input())
42
43 plottedData = [dataPoints[i][inp] / 100 for i in range(len(dataPoints))]
44 step = [dataPoints[i][-1] for i in range(len(dataPoints))]
45
46 # Setup Plot
47 plt.plot(step, plottedData)
48 plt.xlabel("Step Count")
49 plt.ylabel("Average Loss per Step")
50
51 plt.show()
```