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

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 independant 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 it's 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?

Whether I have answered this question or not should be clear. I can specifically measure the Algorithms ability to survive by observing it's actions in given situation, if you Algorithm directs the character into danger on a regular basis, it clearly isn't doing a good job of surviving. If the Algorithm quite clearly isn't good at surviving I have to determine to what extent it solves the problem it's given.

I can determine this by asking more specific questions, proposed below:

1. **Does the Algorithm draw links between specific elements and danger?**
2. **How well does the Algorithm perform with specific tasks?**
3. **If the problem is altered to be simpler does the Algorithm perform better?**
4. **Can I fine tune the Algorithms Parameters to get better results?**

These more specific questions will allow me to determine to what extent the Algorithm can solve the problem. I hope to dive more in depth into answering these in my Evaluation Section later in the project.

The First Question I feel is important because it shows a reasonably high level of understanding of the problem if it can somehow make links between specific parts of its input and determine that they are infact dangerous.

The Second Question may be harder to answer, because I will have to somehow determine just how well it performs doing something like Collecting Items, or Killing Enemies. This can't really be analysed through pure numbers and instead would have to determined through surveying the Algorithm as it works.

The Third Question will definitely be something to test for, I think it will be relatively easy to tell if adjusting the problem shows a noticeable improvement/deterioration in ability to solve the Simulation.

The Fourth Question should be something quite easy to determine if it makes an impact, but overall shall be needed when trying to find the optimal setup for the Algorithm such that it performs its best.

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 havent 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 Q Learning, I believe it has the possibility of solving your problem if not too complex. Because of that you may way 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 when evaluating your project. You could probably save data and plot it using a Graphing Library of some sort, theres bound to be one with your language of choice."

7. What should I focus my Initial Research on?

   "It would be benefical 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 Evaluation of First Interview

My Experts input as part of my Initial Research is useful because it gives me a better understanding of the areas I should target my research on. Below is my thoughts on various parts of the Interview:

- I will definitely focus alot of my research on Neural Networks and Deep Q Learning. Though I will look into other alternatives which I could use.
- User Defined Parameters will definitely be included in the design of my project. I feel as though this would be very helpful for testing, being able to find tune needed Parameters without having to chance them in the code would be useful.
- Perlin Noise sounds like a good method of generating Terrain. I will include this as part of my Research into Algorithm I might utilise.
- This guide for the complexity of my simulation will be useful when designing it. I also like his idea regarding collecting food to stay alive with a hunger bar. Implementing more complex problems like this could be difficult for the Algorithm to solve though.
- Graphing the Average Loss using a Graphing Library would be helpful and support the Evaluation of my Project. It would mean that I'd need to include a Data Logger as part of the design of my project, as well as a library which can plot this Data.
- Pointing me towards learning the complexities behind Forward and Back Propagation is definitely a good starting point as part of my initial research. He also pointed me towards a few helpful resources for learning these two concepts. I will definitely read through these.

1.4.3 Existing Investigations

Crafter

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

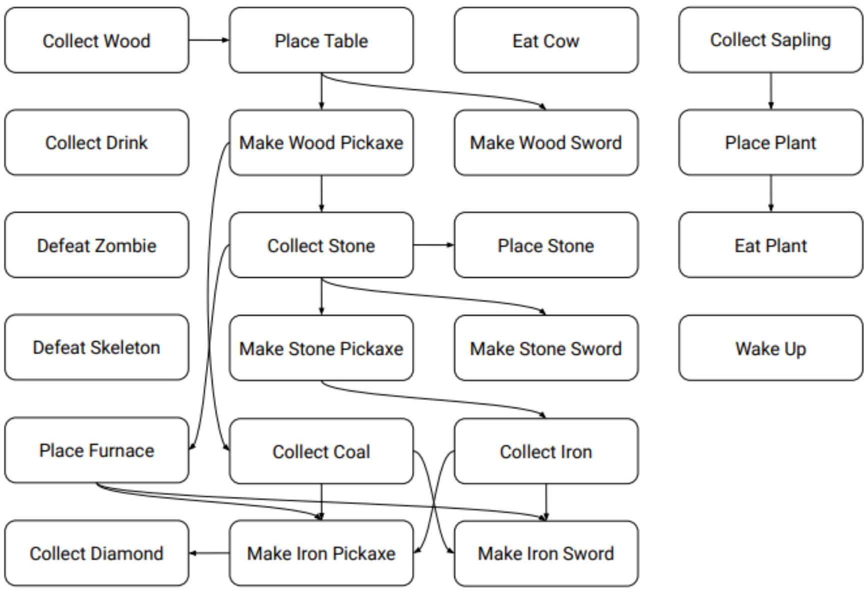
https://github.com/danijar/crafter

Crafter is described to be *"Benchmarking the Spectrum of Agent Capabilities"*, and is utlised 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.

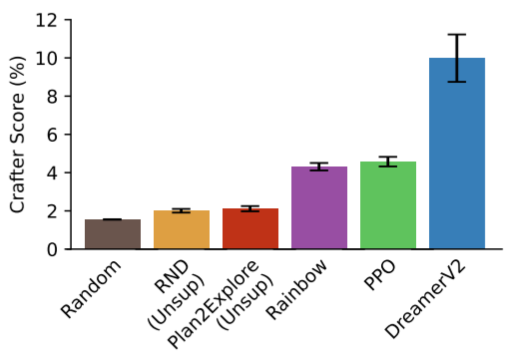
Long-Term reasoning is a complex problem to solve in the context of Machine Learning, current Machine Learning models struggle to deal with this problem. This is dealt with by using algorithms built to mimic "memory". A common implementation of this is Experience Replay which stores states in a queue,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

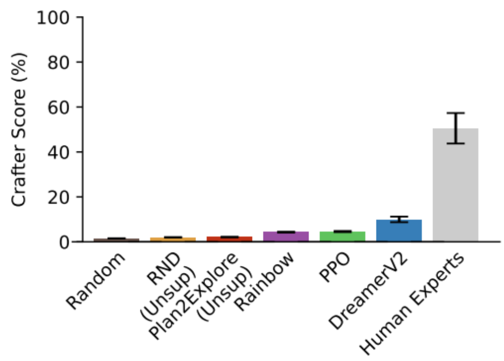


Complex action system as shown in the Paper "Benchmarking the Spectrum of Agent Capabilities"

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.



(a) Success Rate of Algorithms



(b) Comparison Against Human Data

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 plethera of items, enemies, passive animals and more. It has infinite terrain generation, and explicity 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.


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


(b) Example of a Sunken Pirate Ship Structure

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.

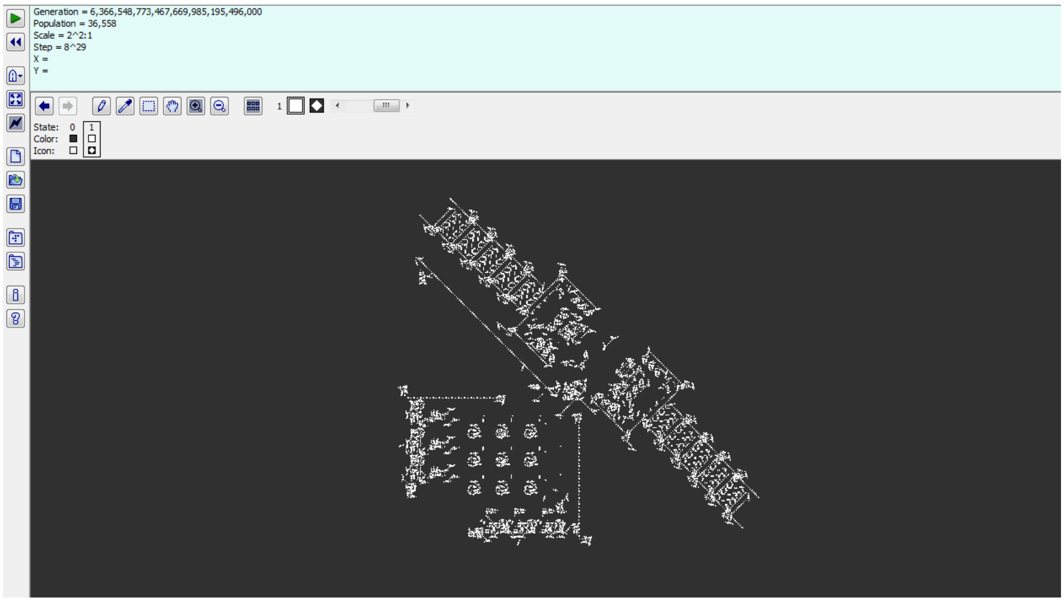
Conway's Game of Life

Conway's Game of Life is whats 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 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 Conways Game of Life

Overall, I think this shows that my simulation doesnt need to have complex rules in order to achieve interesting results. Conway's is formed from 4 simple rules, and yet is Turing complete.

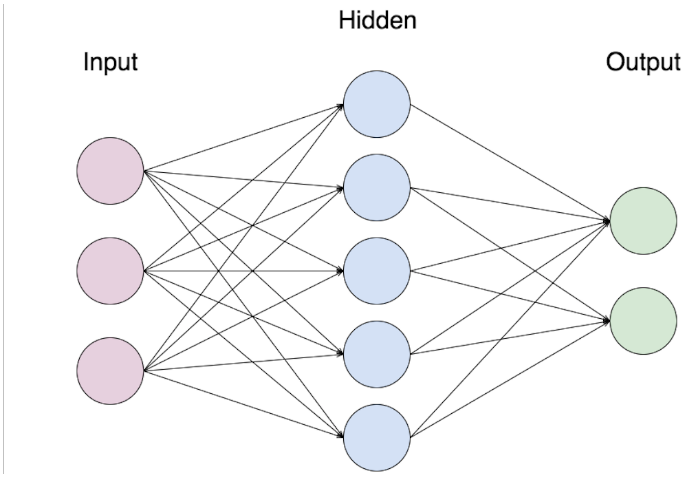
1.4.4 Potential Algorithms

Neural Network and Matrices

If I go down the Deep Q Learning Route, 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: $\{x \in V, N(x) \in V\}$. 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 $[3, 5, 2]$.



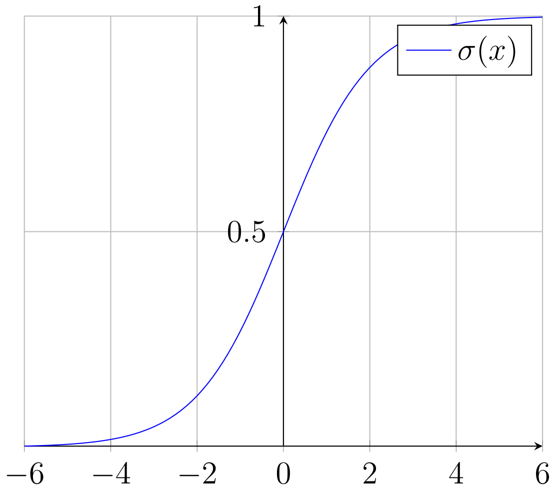
Each connection, otherwise known as an Arc or Edge, has an associated weight. Along with every output of a layer having an associated Bias. These are used to compute the outcome of a network.

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.

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

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

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.

Q-Learning
Another Algorithm I could use for my Machine Learning is Q-Learning. It is the step down from Deep Q Learning, so it is less complex in its Nature. It utilises the Bellman Equation, and something called a Q-Table. A Q table stores every possible state the Agent could be in, and stores values for each action from the State. Every Action for every state has an initial value of 0 and will be updated over time in order to ”learn” the simulation. It is essentially a brute force method to Machine Learning. The greater the Value, the better the Q-Table will consider the action. Below is shown an example of a possible Q-Table I could use for my project:

Q-Table							
		Actions					
		North	East	South	West	Collect	Attack
States:	0	0	0	0	0	0	0
	·	0	0	0	0	0	0
	·	0	0	0	0	0	0
	·	0	0	0	0	0	0
	N	0	0	0	0	0	0

When trained extensively the Q-Table might start to look something like this:

Q-Table							
		Actions					
		North	East	South	West	Collect	Attack
States:	0	-9	-9	6	-12	2	6
	·	-7	6	-7	-14	-14	-3
	·	-1	-20	-10	-14	-5	-4
	·	1	-10	6	-5	-18	-15
	N	-10	-11	-18	5	-9	-2

I feel like the downside to this approach is that it will struggle to generalise to a procedurally generated simulation. If given infinite time and infinite processing power it could theorhetically solve anything, but sadly I dont have that kindof compute power. With the simulation I intend to design there would be to many Possible States for it to ever make a dent in.

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 *realistic* 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.

1.5 Prototype

1.5.1 Prototype Objectives

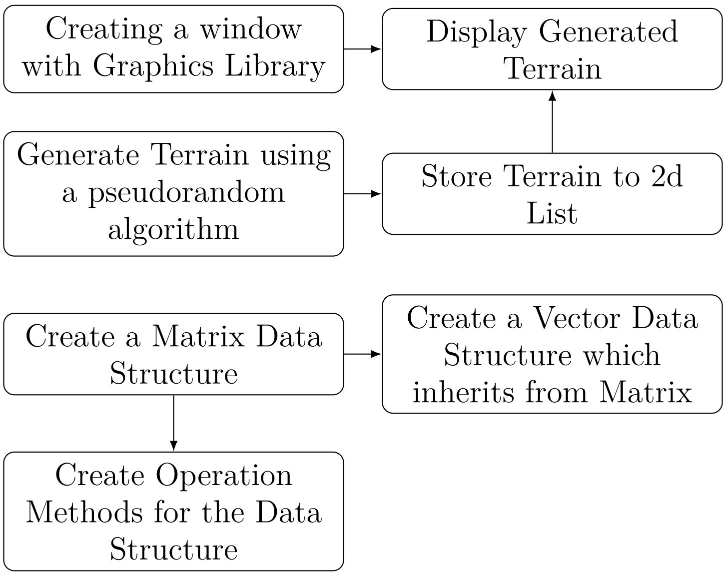
Before starting my Prototype I had to decide upon a short list of objectives I wanted to complete/investigate as part of it. These boiled down to a few things:

- 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:



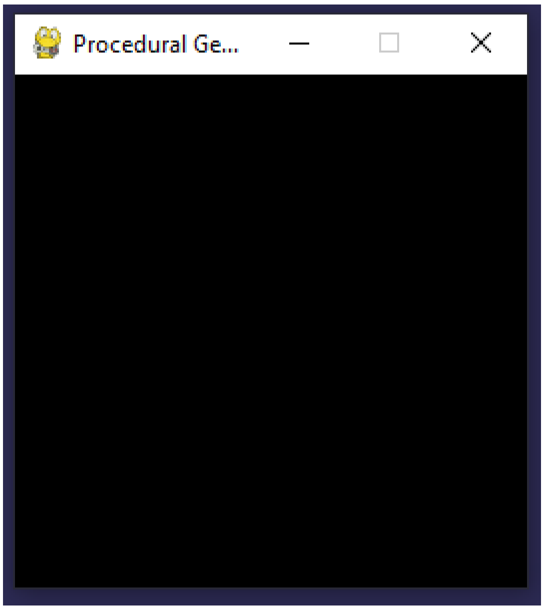
I decided to use Python for developing my Prototype, this seemed like a good fit due to me having lots of experience with the language. Python is a Dynamically Typed and interpreted which makes it versatile for protyping and fast, iterative development.

1.5.2 Terrain Generation and Displaying to Window

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

```python
import pygame

simSize = 128
gridSize = 2

window = pygame.display.set_mode((simSize*gridSize, simSize*gridSize))
pygame.display.set_caption("Procedural Generation")

running = True
while running == True:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      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:

```
def GenerateMap(self, seed):
    random.seed(seed)
    for y in range(0, self.arraySize):
        for x in range(0, self.arraySize):
            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:

$$\begin{aligned} \text{pullValue} &= \begin{cases} \text{upTiles} \times 0.09 & \text{Most Up Tiles} \\ \text{downTiles} \times -0.08 & \text{Most Down Tiles} \\ 0 & \text{Most Neutral Tiles} \end{cases} \\ \text{Value}[x][y] &= \text{pullValue} \end{aligned}$$

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

```python
def UpNeutralDownGen(self):
    dupMap = self.heightArray
    for y in range(0, self.arraySize):
        for x in range(0, self.arraySize):
            up = 0
            down = 0
            neutral = 0
            pointArr = []

            if x != 0 and y != 0:
                pointArr.append(self.heightArray[x - 1][y - 1])
            if x != 0 and y != self.arraySize - 1:
                pointArr.append(self.heightArray[x - 1][y + 1])
            if x != self.arraySize - 1 and y != self.arraySize - 1:
                pointArr.append(self.heightArray[x + 1][y + 1])
            if x != self.arraySize - 1 and y != 0:
                pointArr.append(self.heightArray[x + 1][y - 1])
            if x != 0:
                pointArr.append(self.heightArray[x - 1][y])
            if y != 0:
                pointArr.append(self.heightArray[x][y - 1])
            if x != self.arraySize - 1:
                pointArr.append(self.heightArray[x + 1][y])
            if y != self.arraySize - 1:
                pointArr.append(self.heightArray[x][y + 1])

```

```
            for i in range(len(pointArr)):
                if pointArr[i] >= self.heightArray[x][y] + 0.1:
                    up += 1
                elif pointArr[i] <= self.heightArray[x][y] - 0.1:
                    down += 1
                else:
                    neutral += 1

            if (up > down) and (up > neutral): # Up
                value = 0.09 * up
            elif (down > up) and (down > neutral): # Down
                value = -0.08 * down
            else: # Neutral
                value = 0

            dupMap[x][y] += value
            dupMap[x][y] = self.Clamp(dupMap[x][y], 0, 1)

    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:

$$\begin{aligned} \frac{0.83+0.93+0.64+0.07+0.38+0.21+0.95+0.33+0.94}{9} &= 0.586 \\ Value[x][y] &= 0.586 \end{aligned}$$

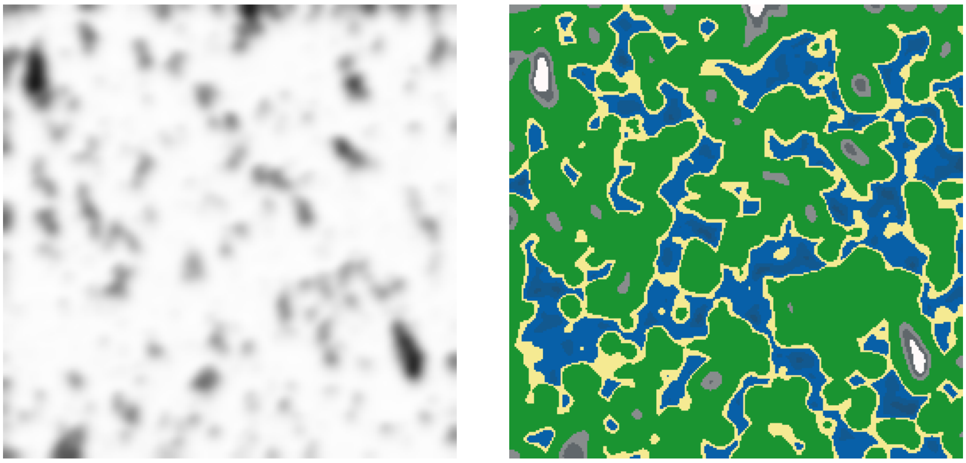
The code for this is shown below:

```
def AverageGen(self):
    dupMap = self.heightArray
    for y in range(0, self.arraySize):
        for x in range(0, self.arraySize):
            total = 0
            count = 0
            if x != 0 and y != 0:
                total += self.heightArray[x - 1][y - 1]
                count += 1
            if x != 0 and y != self.arraySize - 1:
                total += self.heightArray[x - 1][y + 1]
                count += 1
            if x != self.arraySize - 1 and y != self.arraySize - 1:
                total += self.heightArray[x + 1][y + 1]
                count += 1
            if x != self.arraySize - 1 and y != 0:
                total += self.heightArray[x + 1][y - 1]
                count += 1
```

```
            if x != 0:
                total += self.heightArray[x - 1][y]
                count += 1
            if y != 0:
                total += self.heightArray[x][y - 1]
                count += 1
            if x != self.arraySize - 1:
                total += self.heightArray[x + 1][y]
                count += 1
            if y != self.arraySize - 1:
                total += self.heightArray[x][y + 1]
                count += 1

            dupMap[x][y] = total / count
    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:

```
@staticmethod
def MatrixAddSubtract(m1, m2, subtract = False):
    m1Dims = m1.Dimensions()
    newMat = Matrix(m1Dims[0], m1Dims[1])
    for y in range(0, m1Dims[0]):
        for x in range(0, m1Dims[1]):
            if subtract:
                newMat.matrixArr[y][x] = m1.Val()[y][x] - m2.Val()[y][x]
            else:
                newMat.matrixArr[y][x] = m1.Val()[y][x] + m2.Val()[y][x]
    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 multiples 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:

```
@staticmethod
def ScalarMultiply(s, m1):
    m1Dims = m1.Dimensions()
    newMat = Matrix(m1Dims[0], m2Dims[1])
    for y in range(0, m1Dims[0]):
        for x in range(0, m1Dims[1]):
            newMat.matrixArr[y][x] = m1.matrixArr[y][x] * s

@staticmethod
def MatrixMultiply(m1, m2):
    m1Dims = m1.Dimensions()
    m2Dims = m2.Dimensions()
    newMat = Matrix(m1Dims[0], m2Dims[1])
    for row in range(0, m1Dims[1]):
        subRow = m1.Val()[row][0:m1Dims[1]]
        for col in range(0, m2Dims[1]):
            subCol = []
            for i in range(0, m1Dims[0]):
                print(i)
                subCol.append(m2.Val()[i][col])
            total = 0
            for x in range(0, len(subRow)):
                total += subRow[x] * subCol[x]
            newMat.matrixArr[row][col] = total
    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:

```
def SubMatrixList(self, rowList, colList):
    newMat = Matrix(self.Dimensions()[0] - len(rowList),self.Dimensions()[1] - len(colList))
    xoffset = 0
    yoffset = 0
    yRowList = []

    for y in range(0, self.Dimensions()[0]):
        for x in range(0, self.Dimensions()[1]):
            if x in colList and y in rowList:
                xoffset += 1
                yoffset += 1
                continue
            elif x in colList:
                xoffset += 1
                continue
            elif y in rowList and y not in yRowList:
                yoffset += 1
                yRowList.append(y)
                continue
            else:
                newMat.matrixArr[y - yoffset][x - xoffset] = self.matrixArr[y][x]
        xoffset = 0
    return newMat

@staticmethod
def Determinant(m):
    dims = m.Dimensions()
    if dims[1] <= 2:
        det = (m.matrixArr[0][0] * m.matrixArr[1][1]) - (m.matrixArr[0][1] * m.matrixArr[1][0])
        return (det)
    elif dims[1] != 2:
        det = 0
        subtract = False
        tempMat = m.SubMatrixList([0],[])
        for i in range(0, dims[1]):
            subMat = None
            subMat = m.SubMatrixList([0],[i])
            if subtract == False:
                det += m.matrixArr[0][i] * Matrix.Determinant(subMat)
                subtract = True
            elif subtract == True:
                det -= m.matrixArr[0][i] * Matrix.Determinant(subMat)
```

```
                subtract = False
        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.

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

   The written code is shown below:

```
@staticmethod
def DotProduct(v1,v2):
    total = 0
    for i in range(v1.Dimensions()[0]):
        total += v1.Val()[i][0] * v2.Val()[i][0]
    return total
```

1.5.5 Prototype Evaluation

Overall I am happy with my prototype, though I feel like it needs improvement. I did meet my objectives for my prototype but there are things I want to change for my Technical Solution. 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 also be rewritten to be more efficient, my implementation of the Operations is not the best. 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 Further Research

1.6.1 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 its 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. Is an Object Orientated Network a good approach?

"Implmenting your Network using a Class hierarchy would allow you to organise your code nicely, passing objects between methods. With a Dual Neural Network you could create two instances of a Network class using them as your Main and Target Network. This would also gain your marks for coding style."

4. 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."

5. What type of Reward Structure should I use?

   "As far as I'm aware there are two types of Reward Structures, Sparse and Dense. I think that Dense 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. Say if you want to motivate exploration I feel like dense would be appropriate."

6. What type of pathfinding should I use for the Enemies in my Simulation?

   "I don't think it would be necessary to implement any complex algorithm for your Enemies Pathfinding, it will only eat into your performance when training the Network. But if you do want to implement a complex algorithm, Dijkstras or A* Search would be appropriate."

1.6.2 Evaluation of Second Interview

My Experts feedback has been useful in this case. Below is my thoughts on various parts of the Interview:

- I am in agreement with him about my prototype, it really has alot of improvements to be made. Shaun mentioned Operator overloading which I was unaware Python could utilise at the time of creating my Matrix Class, I will definitily use this in my Technical Solution.
- I will definitely be implementing a Dual Neural Network then. He shared his concerns about it's ability to solve complex problems of this Nature. Due to the nature of my project, this will be fine as I'll still be able to analyse the Networks Results. The results will hopefully show a declining trend in Network Loss.
- I will be using an Object Orientated Model for my Program so I feel like this is the best way forward. His suggestion about the two instances of Network for Main and Target will be useful.
- Encouraging exploration is something I definitely want to do so I will try to implement a Dense Reward Structure.
- I didn't have any plans to implement a complex path finding algorithm so this is helpful to see we're in agreement.

1.6.3 Research Material

As part of my Further Research, I did more research relating to specific parts of my project. These are listed below along with descriptions about their contents and what I've gained from them:

Playing Atari with Deep Reinforcement Learning - DeepMind
This paper contained helpful information regarding the Deep Q Learning Algorithm, allowing me to gain a deeper understanding of how it works. The main goal of this Paper was to train an Algorithm to play Atari Games through using raw pixel data. They manage to achieve relatively good results, with the Average Reward per Epoch and Average Action Value showing positive trends for the tested games.

Revisiting Fundamentals of Experience Replay - DeepMind
This Paper defines the Experience Replay Algorithm in use with Deep Q Networks. It allowed me to gain an insightful understanding into the motivations behind how the Algorithm works. It also outlines a process for Prioritized Experience Replay, but I did not end up using this as part of my project.

Benchmarking the Spectrum of Agent Capabilities - Google Research
I read this Paper as part of my Initial Research. It was interesting to see something so similar to my Project being researched at a higher level. The problems they struggled which I outlined in my Existing Investigations Section informed some of my simulation design, in an attempt to Mitigate the problems they face.

Fast Poisson Disc Sampling in Arbitry Dimensions - Robert Bridson
This Paper outlines the Algorithm used for Poisson Disc Sampling in detail. There is not much else to it. I saved it for later for when I come to implementing it in my Technical Solution.

Neural Networks - 3Blue1Brown
This is a helpful video series designed to explain Neural Networks from a ground up point of view. It goes through all the Maths in a comprehensive way, which I found very helpful when trying to get my head around Back Propagation. Having a video series I can rewatch over and over was helpful in this process.

Mathematics of Backpropagation - Brian Dolhansky
This article about Back Propagation was very helpful when trying to find a general form for Matrix Back Propagation. Its lays out Neural Networks from an easy to follow standpoint, making it easier for me to find my own General Matrix Form, shown in Modelling of the Problem.

1.7 Objectives

Taking into account my Prototype and Interviews, 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.

1. The Program must have User Input
   - 1.1 Before Launching the Program the User should be able to adjust individual Parameters of the Program
   - 1.2 Examples of such Parameters are:
     - 1.2.1 World Size
     - 1.2.2 Tile Colour Values
     - 1.2.3 Neural Network Layer Size
   - 1.3 These Parameters should be stored in a User Readable File/Plain Text

1.4 These Parameters should be checked against Ranges before being used in the Program
1.5 These Ranges should be specified in a seperate File
1.6 An Error message should be displayed if a Parameter is Out of Range
1.7 The User should be prompted to load previous Training Data from a File

2. The Program must have Graphical Output

2.1 After User input is complete a Graphical Window should open
2.2 This Window should display the current state of the simulation
2.3 The Agent should be displayed to the Window
2.4 The Enemies should be displayed to the Window
2.5 The Objects should be displayed to the Window
2.6 The Window should display each element with it's correct Colour
2.7 When enabled a Debug Side Bar should be displayed
2.8 The Debug Side Bar should the Neural Networks Activation Values

3. The Program must contain a Matrix Implementation

3.1 Data for the Matrix must be stored in a 2d List/Array
3.2 A Property must exist for the order of the Matrix
3.3 Multiple initialisation Methods must be possible
3.3.1 Creation using a Tuple of Integers
3.3.2 Creation using an existing 2d List
3.3.3 Creation using an existing 1d List
3.4 It must be possible to fill a Matrix with Randomised Values
3.5 It must be possible to create the Identity Matrix of N Size
3.6 There should be a way to print a Matrix to the Console
3.7 There should be Methods for Standard Matrix Operations
3.7.1 Addition
3.7.2 Subtraction
3.7.3 Multiplication
3.7.4 Scalar Multiplication
3.7.5 Hadamard Product
3.7.6 Transpose
3.7.7 Sum
3.8 Most of these Operations should utilise Operator Overloading where possible
3.9 The Operations should be implemented utilising Efficient Algorithms (Minimised Time Complexity)
3.10 Appropriate Error messages should be in place when utilising Matrices incorrectly

4. The Program must contain an Agent and Simulation Environment

4.1 The Simulation Environment must be stored in a WorldMap Object
4.2 The Simulation Environment must be Procedurally Generated
4.2.1 The Simulation Environment must be generated using a Seed
4.2.2 The Terrain must be Generated utilising the Perlin Noise Algorithm

- 4.2.3 The Terrain Data must be Stored in Tile Objects
- 4.2.4 Each Tile Object must be assigned Tile Type based on its Height Value
- 4.2.5 Each Tile Type must use a Colour Specified by the User
- 4.2.6 Objects must be placed around the Environment for the Agent to Collect
- 4.2.7 The Objects must be placed utilising the Poisson Disc Sampling Algorithm
- 4.2.8 Enemies must be randomnly placed around the Environment at the start of a World
- 4.2.9 The Enemies must pathfind towards the Agent

- 4.3 The Simulation must have Finite State Transitions
- 4.4 The Agent must be spawned into the Simulation
- 4.5 The Agent must be controlled by the Machine Learning Algorithm
- 4.6 The Agent must have a Finite Action Set
- 4.7 The Agent must have a Position which can be altered by it's Actions
- 4.8 The Agent must be able to Pickup Items
- 4.9 Any Collected Items must be stored in the Agents Inventory
- 4.10 The Agent must be able to Attack Enemies
- 4.11 The Agent must be able to be killed by the Enemies
- 4.12 The Agent should be killed when it traverses into Water
- 4.13 When the Agent is killed the Simulation should Reset
- 4.14 The Agent must have a Reward Structure
  - 4.14.1 The Agent must gain Reward from doing specific things
  - 4.14.2 Reward must be gained from exploration
  - 4.14.3 Reward must be gained from Killed an Enemy
  - 4.14.4 Reward must be gained from Collecting an Item
  - 4.14.5 Reward must be lost from Dying
  - 4.14.6 Reward must be lost from Failing an Attack
- 4.15 When the Simulation is reset the Terrain should be Procedurally Generated again
- 4.16 When the Simulation is reset the Objects should be placed again
- 4.17 When the Simulation is reset the Enemies should be spawned again
- 4.18 When the Simulation is reset the Agent should be spawned again
- 4.19 The Agent must be able to Sample its Environment for Tile Data
- 4.20 This Tile Data must be able to be converted into Grayscale Values
- 4.21 These Grayscale Values must be used as the Neural Networks Input

5. The Program must contain a Dual Neural Network

- 5.1 The Dual Neural Network must be formed from two instances of a Neural Network Class
  - 5.1.1 The Neural Network must be implemented utilising an Object Orientated Model
  - 5.1.2 The Neural Network must contain Forward Propagation
  - 5.1.3 The Neural Network must utilise the Bellman Equation to find Expected Values
  - 5.1.4 The Expected Values must be used when Calculating the Loss of the Network
  - 5.1.5 The Neural Network must contain Backwards Propagation
  - 5.1.6 The Back Propagation must utilise the Calculated Loss when calculating the Error Signal

5.1.7 The Forward and Back Propagation must be Implemented utilising Matrix Operations
5.1.8 The Neural Network must be able to utilise Activation Functions

5.2 The Target Network must be updated intermittently
5.3 The Neural Network must be Trainable by inputting collected Grayscale Values
5.4 The Agent must collect these Grayscale Values
5.5 The Neural Network should instruct the Agents Actions
5.6 The Neural Network should utilise the Epsilon Greedy decision process
5.7 The Neural Network should utilise the SoftMax Function when picking an Action
5.8 The Neural Network should utilise Experience Replay
5.9 Each State should be stored in an Experience Object
5.10 Experience Replay should be performed intermittently
5.11 The Training Data must be able to be Saved as .dqn Files
5.12 Experience Replay States should be stored in a Double Ended Queue

6. The Program must contain Activation Functions

6.1 The Activation Functions must be used within the Dual Neural Network
6.2 An Activation Function must be defined as an Abstract Base Class
6.3 An Activtion Function must have a Normal and Derivative Method
6.4 An Activation Function must take a Vector as an Input
6.5 An Activation Function must return a Vector as an Output
6.6 There must be standard Activation Functions implemented

6.6.1 Sigmoid
6.6.2 TanH
6.6.3 ReLu
6.6.4 Leaky ReLu

7. The Program must contain a Data Logger

7.1 The Data Logger must use a specific Structure
7.2 When adding a Data Point it must match the Data Loggers Structure
7.3 If it does not it should display an Error Message
7.4 The Data Points should be saved to a .data File
7.5 The Data Logger must contain a Heap Sort to find minimum and maximum values
7.6 The Heap Sort must allow sorting via an index of the Data Point
7.7 The Heap Sort must utilise an Efficient Algorithm (Minimised Time Complexity)

8. The Program must contain a Script to Graph Data

8.1 The Script must present the User options for which .data file to plot
8.2 The Script must present the User options for which part of the Data to Plot
8.3 The Script must display a graph of the data

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 x 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.

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

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

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.

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

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.

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

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.

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

Such that

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

Matrix Scalar Multiplication

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

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

Matrix Hadamard Product

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

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

Matrix Transpose

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

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

1.8.2 Forward Propagation

Overview

Forward Propgation 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:

$$\begin{aligned} a_i^{(L)} &= \text{The Activation Value for the } i^{th} \text{ Node in the } L^{th} \text{ Layer} \\ z_i^{(L)} &= \text{The Pre-Activation Value for the } i^{th} \text{ Node in the } L^{th} \text{ Layer} \\ w_{m,n}^{(L)} &= \text{The Weight between node } n \to m \text{ from the } L^{th} \text{ to the } (L+1)^{th} \\ b_i^{(L)} &= \text{The Bias Value for the } i^{th} \text{ Node in the } L^{th} \text{ Layer} \end{aligned}$$

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.

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

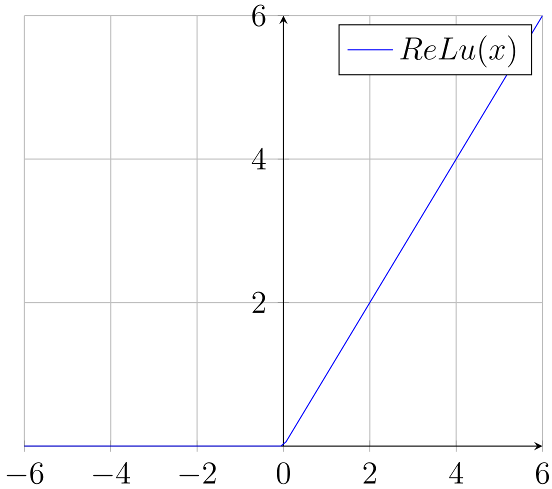
$$Z^{(L)} = \begin{bmatrix} z_0^{(L)} \\ z_1^{(L)} \\ \vdots \\ z_n^{(L)} \end{bmatrix} = \begin{bmatrix} w_{0,0}^{(L-1)} & w_{0,1}^{(L-1)} & \dots & w_{0,m}^{(L-1)} \\ w_{1,0}^{(L-1)} & w_{1,1}^{(L-1)} & \dots & w_{1,m}^{(L-1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,0}^{(L-1)} & w_{n,1}^{(L-1)}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

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:

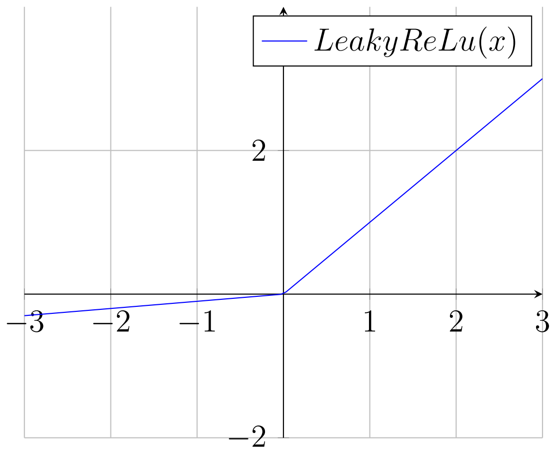
ReLu

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



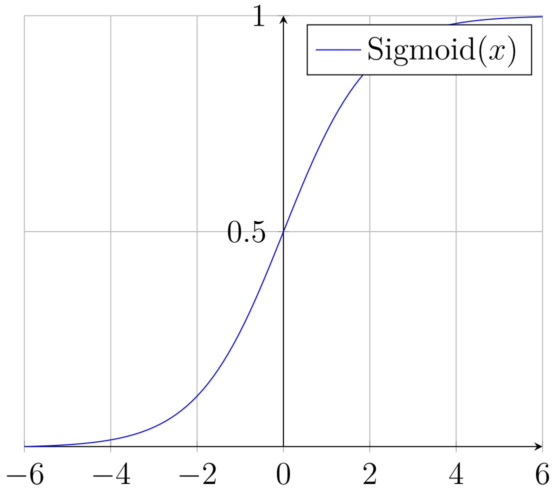
Leaky ReLu

$$\text{LeakyReLu}(x) = \begin{cases} x < 0 & 0.1 \times x \\ x > 0 & x \end{cases}$$



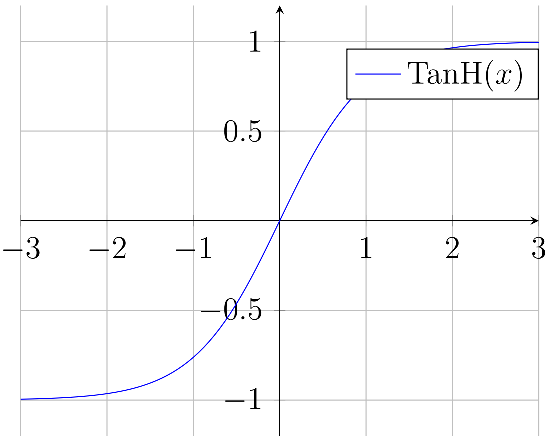
Sigmoid

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



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.

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

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

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 fininite 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.

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

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 $f(x) = g_1(g_2(\ldots(g_n(x))))$. By this rule we can represent the derivative as a Series of Derivatives Multiplied together:

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

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 seperately 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:

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

$$\text{When subjected to: } x_1 = T(s_0, a_0)$$

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.:

$$Q(s_t, a_t) = R(s_t, a_t) + \gamma \cdot \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

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

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.

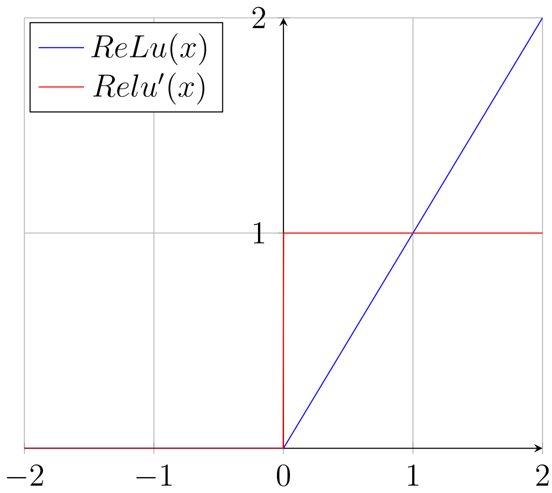
Differentiating Activation Functions

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

The ReLu Derivative:

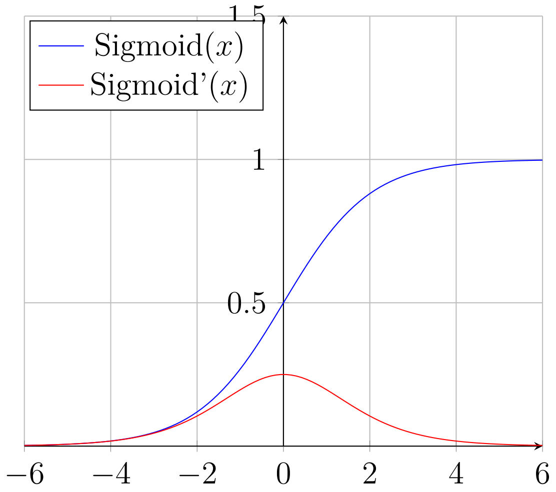
$$\text{ReLu}(x) = \begin{cases} 0 & x < 0 \\ x & x > 0 \end{cases}$$

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



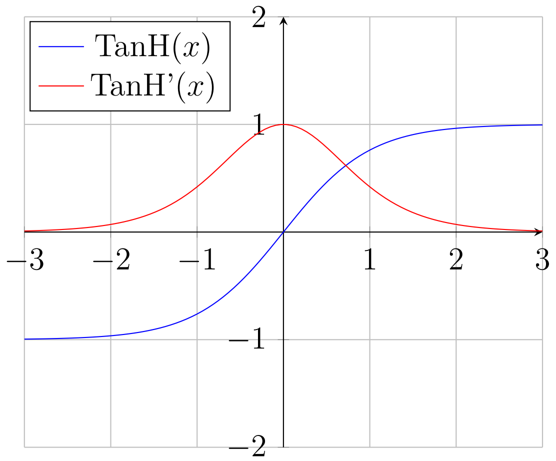
The Sigmoid Function Derivative:

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

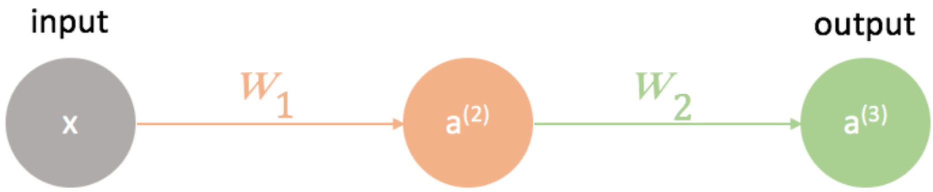


The TanH Derivative:

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



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 $\dfrac{\partial c}{\partial a_3}$.

$$\begin{aligned} c &= \frac{1}{2}\cdot(y-a_3)^2 \\ \frac{\partial c}{\partial a_3} &= y - a_3 \end{aligned}$$

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

$$\begin{aligned} a_3 &= \frac{e^{z_3}-e^{-z_3}}{e^{z_3}+e^{-z_3}} \\ &= \tanh(z_3) \\ \frac{\partial a_3}{\partial z_3} &= 1-\tanh^2(a_3) \end{aligned}$$

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

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

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

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

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

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

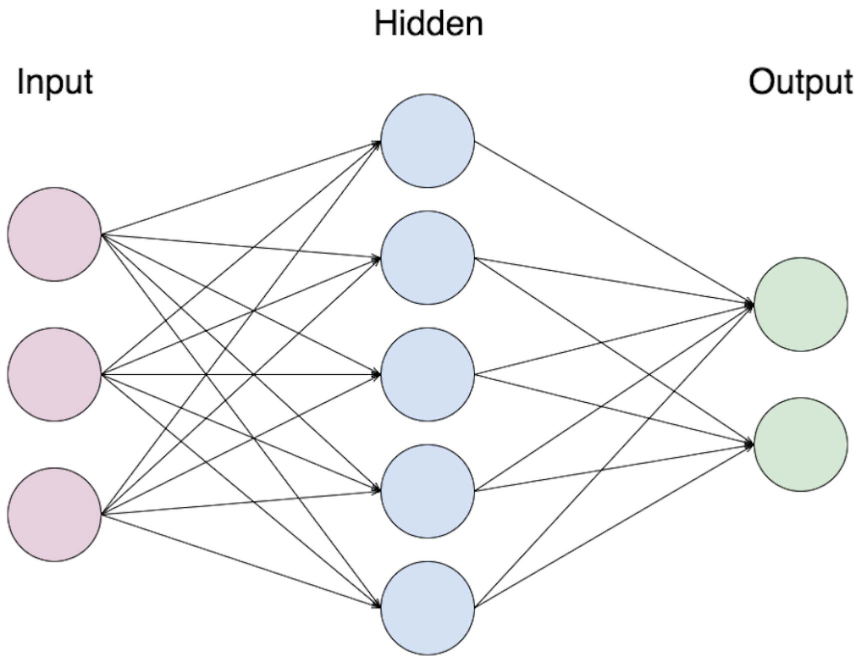
It is however similar to our original chain, the only new derivative is $\dfrac{\partial z_3}{\partial a_2}$. Which is simply w_2, leaving us with the following derivative:

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

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

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

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 $1, 2, \ldots, n$:

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

This should be all that is needed to perform Back Propagation, but we can convert this into it's Matrix form reduce the required operations. Below is defined the needed Matrices which store elements of the Network.

$$Z^{(L)} = \begin{bmatrix} z_0^{(L)} \\ z_1^{(L)} \\ \vdots \\ z_n^{(L)} \end{bmatrix} \qquad W^{(L)} = \begin{bmatrix} w_{0,0}^{(L-1)} & w_{0,1}^{(L-1)} & \dots & w_{0,m}^{(L-1)} \\ w_{1,0}^{(L-1)} & w_{1,1}^{(L-1)} & \dots & w_{1,m}^{(L-1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,0}^{(L-1)} & w_{n,1}^{(L-1)} & \dots & w_{n,m}^{(L-1)} \end{bmatrix}$$

$$A^{(L)} = \begin{bmatrix} a_0^{(L-1)} \\ a_1^{(L-1)} \\ \vdots \\ a_n^{(L-1)} \end{bmatrix} \qquad B^{(L)} = \begin{bmatrix} b_0^{(L)} \\ b_1^{(L)} \\ \vdots \\ b_n^{(L)} \end{bmatrix}$$

Using these Matrices we then calcuate the Weight and Bias derivatives. The Layer N is the final layer, C Networks output with the Loss Functions derivative applied. This form is shown below:

$$\begin{aligned} \delta^{(N)} &= C \odot \sigma' \cdot Z^{(L)} \\ \delta^{(L)} &= ((W^{(L)})^T \cdot \delta^{(L+1)}) \odot \sigma' \cdot Z^{(L)} \\ \beta^{(L)} &= \delta^{(L)} \\ \omega^{(L)} &= \delta^{(L)} \cdot (A^{(L)})^T \end{aligned}$$

2 Design

2.1 Programming Language and Libraries

I chose Python for my chosen Programming Language, it's very versatile and I have lots of experience with the language already. It's great for rapid prototyping which I feel is necessary for a project of this scale. I already used it for my prototype so I will be able to reuse some parts of my previous code base. Such as my Matrix and WorldMap Class'.

Below is a list of key libraries I will be using in my project:

Pygame

Pygame is a highly customizable and well developed binding of *Simple DirectMedia Layer* (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.

I will be using Pygame to graphically display the Environment I create as part of my Technical Solution. This will be done in a similar way to my prototype, displaying each tile as a specified colour. Pygame is purely a method to graphically output the current state of the simulation.

This will be used to fulfil the whole of **Objective 2**

Json

The ability to load Json Formatted Files is a key part of my User Input and overall Technical Solution. The Json Library in Python allows this with relative ease, along with saving Json data where needed. I will be using Json files to store the User Inputted Parameters to the program. These parameters will be things like the size of the Simulation, and Neural Network Structure.

This will be used to fulfil parts of **Objective 1**, more specifically **Objectives 1.1, 1.3, 1.5**

Pickle

Pickle will be used to write Binary Files with Python. You can use it to save objects, such as classes or lists, and load them. I will be using pickle to store the Matrix Class when saving the Training Progress of the Neural Network. Each Weight and Bias will need to be stored in order to resume the exact training position of the Network. I will also be storing states for Experience Replay, and the Data Points for plotting Training Data.

This will be used to fulfil **Objectives 5.11, 7.4, 8.1**

MatPlotLib

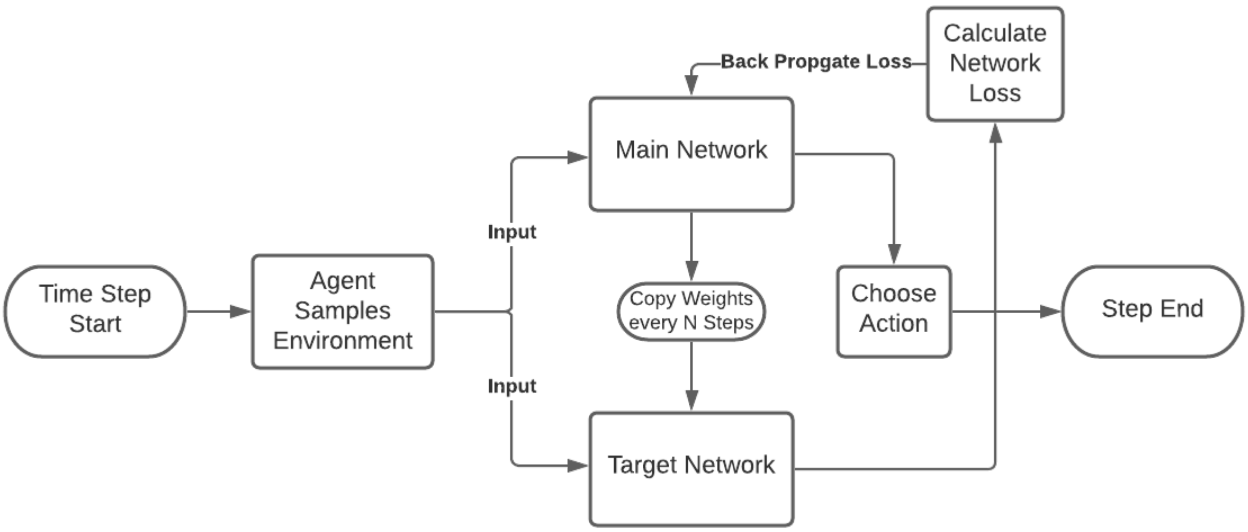
MatPlotLib is a simple way to visualise Numerical data. You can very easily plot graphs from a set of data points. With my Technical Solution I intend to load data previously stored with Pickle, and plot it with this Library.

This will be used to fulfil **Objective 8.3**

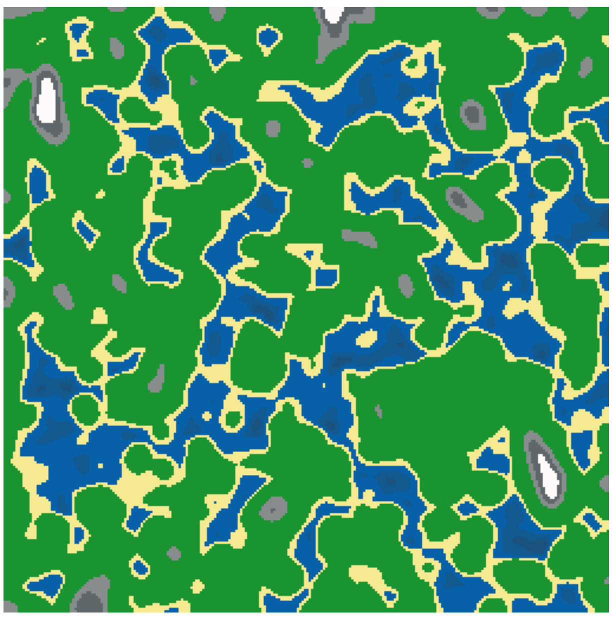
2.2 High Level Overview

The main purpose of the project is to answer my investigations question. I'm answering this question by developing a program which simulates an Environment in which a Machine Learning Algorithm can Explore and Interact with. The user will be able to alter the parameters of this Machine Learning Algorithm and Simulation in order to test different aspects. This will be done through a Json File in which the listed parameters will have specified ranges they must be between. The Descriptions and Ranges of these Parameters are shown under the File Structure Section.

The Machine Learning Algorithm will be Deep Q Learning, utilising a Dual Neural Network at its core. A Dual Neural Network is formed from 2 Neural Networks, a Main and Target. Within Deep Q Learning we are updating a guess with a guess, this leaves us with instability. To solve this, the Target Network is a copy of the Main Network, made every N Steps, and is used to inform the Bellman Equation (mentioned in Modelling of the Problem) when calculating Expected Values in the Loss Function. Below is shown a diagram of how a Dual Neural Network functions:



The simulated environment will be procedurally generated using Perlin Noise and Poisson Disc Sampling. Perlin Noise will generate a Height Map of values, these values will then get mapped to colour bands (which are specified by the User). Poisson Disc Sampling will be used to place Items around the Environment which the Agent will be able to interact with through an Action. The Terrain and Items will be displayed to the screen via a Window, similar to my prototype like this:

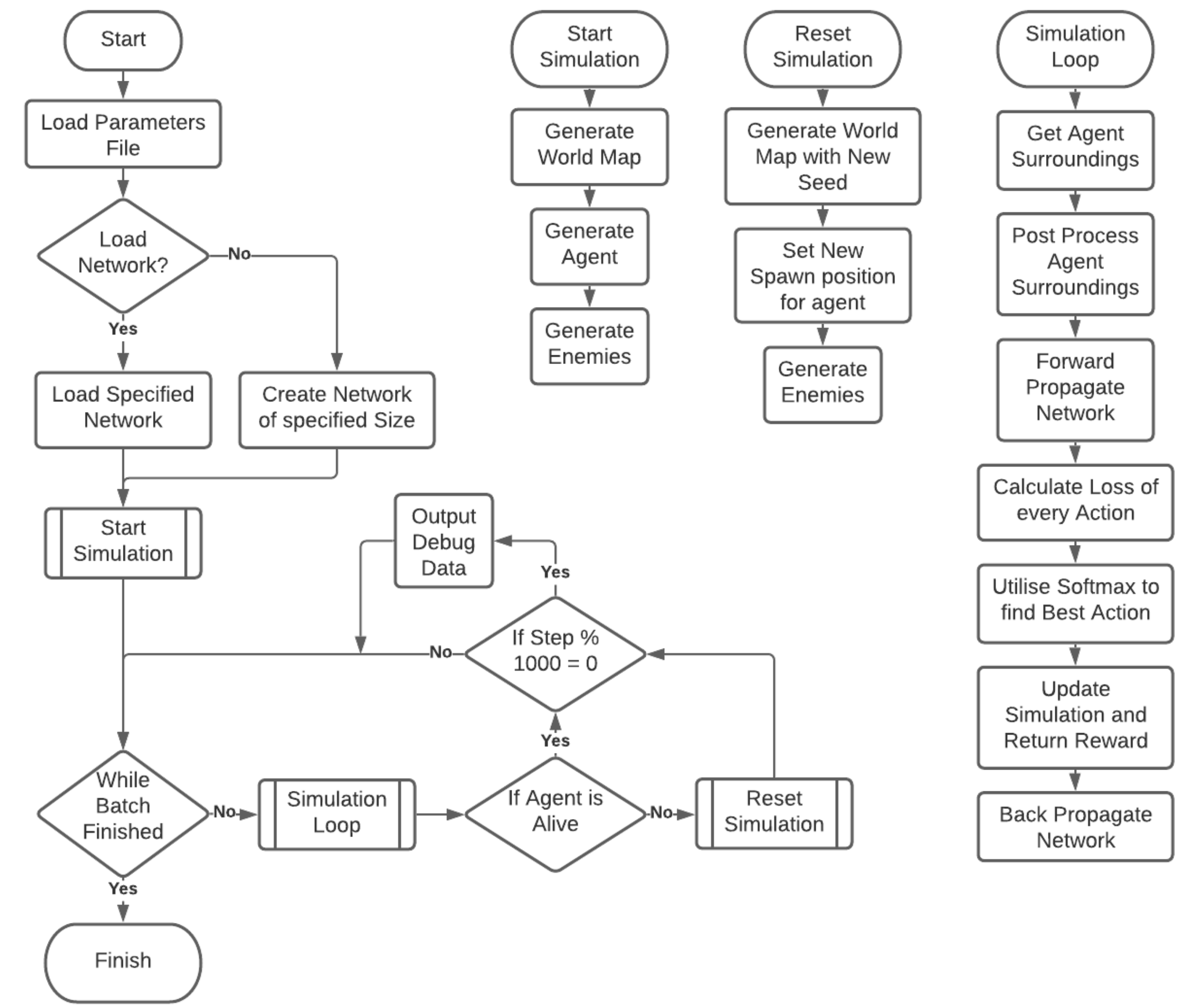


Terrain Generated by my Prototype

Within the simulated environment are generated Enemies which pathfind towards the Agent in an attempt to hinder the Agents surivival. This can be done using a simple pathfinding algorithm. If these enemies are ever to touch Water, they will die. The Agent will be a character controlled within the environment by the Deep Q Learning Algorithm, which has a specific set of Actions. Upon picking an action the Agent will be rewarded or penalised depending on which action is picked in which state. If the Agent is ever in a Water Tile or within an Enemy, the Agent will be penalised, and the environment will be generated again. To pick the Agents Action the surrounding Terrain is sampled for Grayscale Colour Values. These Grayscale values are then passed into the Main Neural Network to inform the decision. With the use of the SoftMax Logistics Function, we can generate a probability distribution from the Neural Network outputs.

With the use of a training method called Epsilon Greedy we can balance Exploration and Exploitation. Epsilon is defined as a value at the start of the Training Session, and is slowly multiplied by a regression value each step. When picking an action a Random Number is generated between 0 and 1, and then compared with the Epsilon Value. If the random value is less than Epsilon we pick a random Action, Exploration. If the random value is greater than Epsilon we use an informed decision, Exploitation.

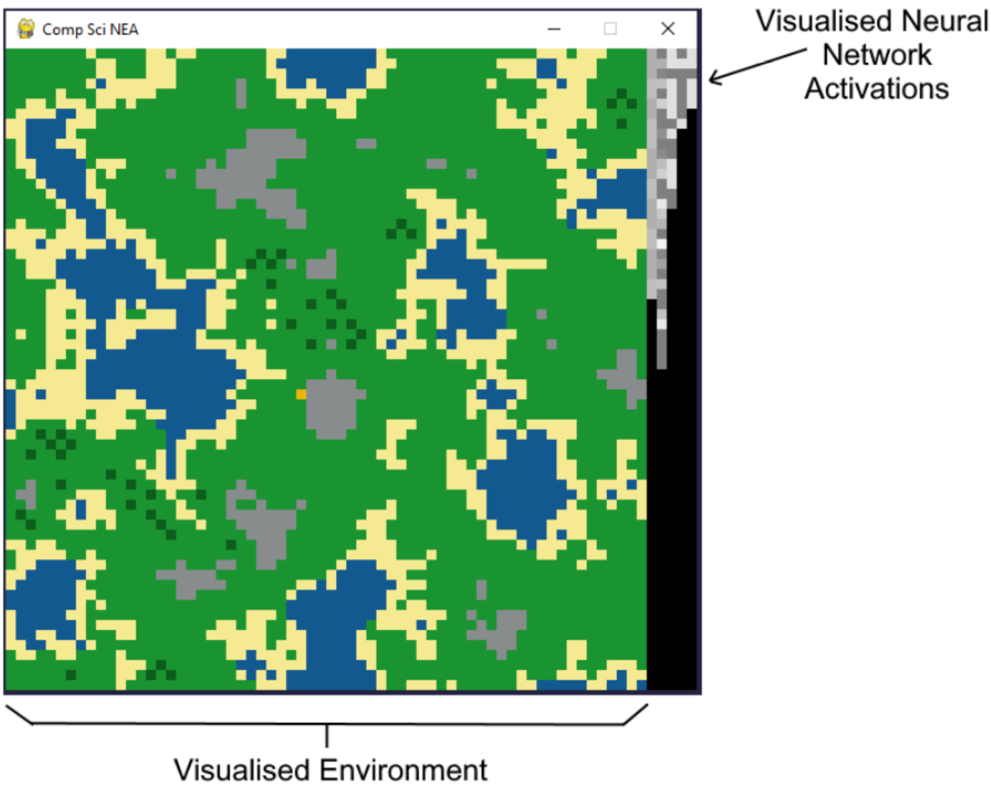
Below is shown a simplified Flow Chart of the whole process:



2.3 User Interface/Graphical Output Design

The Graphical Output for the Program will be a display the current environment as a Grid of Coloured Tiles. I have previously implemented the display of a Grid of Tiles in my Prototype, but I want to expand this to include a Debug Menu at the side.

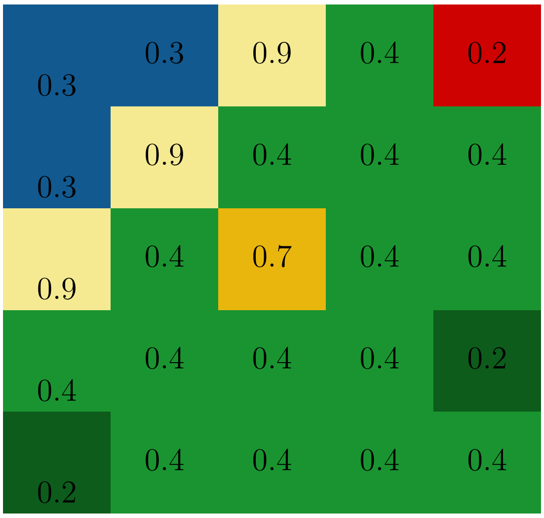
The Debug Menu will be enabled through the Json File, and will display the Neural Network Activations as grayscale values. It will scale to the size of the Network, and will be helpful for debugging. Below is shown a UI Screenshot from the final build:



2.4 Agent Interaction and Reward Design

The Agents interaction with the Algorithm and Simulation is very important. If designed incorrectly the Algorithm will be unable to grasp the Simulation Properly.

The Agent will sample the surrounding environment for Grayscale colour values of each Tile. This Grayscale data will then be converted into a Vector to utilise it as the Neural Networks input. This works well because Grayscales Values are a value from $0 \rightarrow 1$, values like this are often used as Neural Network inputs. Below I have marked the Grayscale values (1 Decimal Place) on each of the colurs from a sample situation. The colour conversion will be done using a simple algorithm which is shown later in Design under Description of Algorithms.



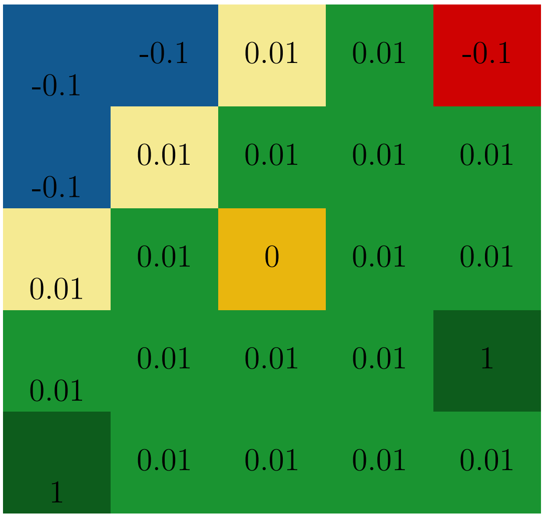
The Agent will have a specific set of actions it can use at any time to interact with the Environment. These actions may be good or bad given each individual situation, that is for the Algorithm to try and decipher through the gains and loss' of Reward. Below is shown a table of actions the Agent will utilise. I designed this action set to be relatively simple as to keep the complexity of the Simulation as little as possible.

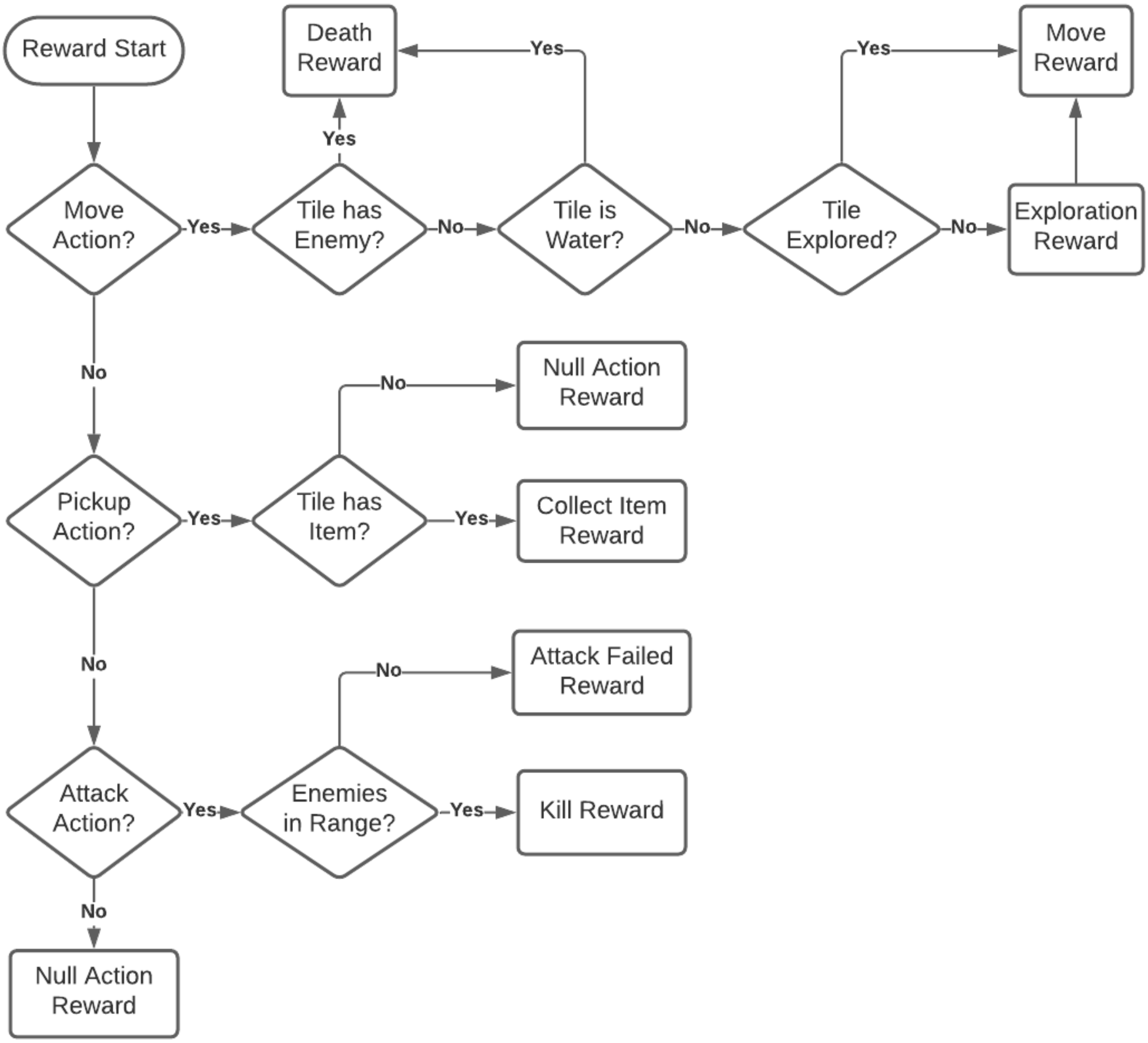
Action No.	Action Name	Action Description
1	Move Up	Agent Moves Up one Tile
2	Move Right	Agent Moves Right one Tile
3	Move Down	Agent Moves Down one Tile
4	Move Left	Agent Moves Left one Tile
5	Pickup Item	The Item on the same Tile as the Agent is collected
6	Attack	Enemies within a radius of the Agent are attacked
7	Noop	Null Action / No Action performed

The Reward Structure for a given Situation is very important, if the Agent is being rewarded incorrectly this could lead to drastic issues with training and calculating expected values. I decided to focus my Reward Structure on Exploration and Motivating the Agent to collect Items while also rewarding successfully eliminating Enemies. This was based off of feedback which was given to me by my Expert in my second Interview. Below is shown a table of the individual Reward values I assigned when initially designing my Project.

Reward Name	Value	Reward Description
Explore Reward	0.01	Given to the Agent when it enters a previously unvisited Tile
Collect Item Reward	1	Given to the Agent when it collect an Item
Attack Reward	0.5	Given to the Agent upon a successfull Attack
Attack Failed Reward	-0.1	Given to the Agent when an Attack has failed
Death Reward	-1	Given to the Agent when it is killed

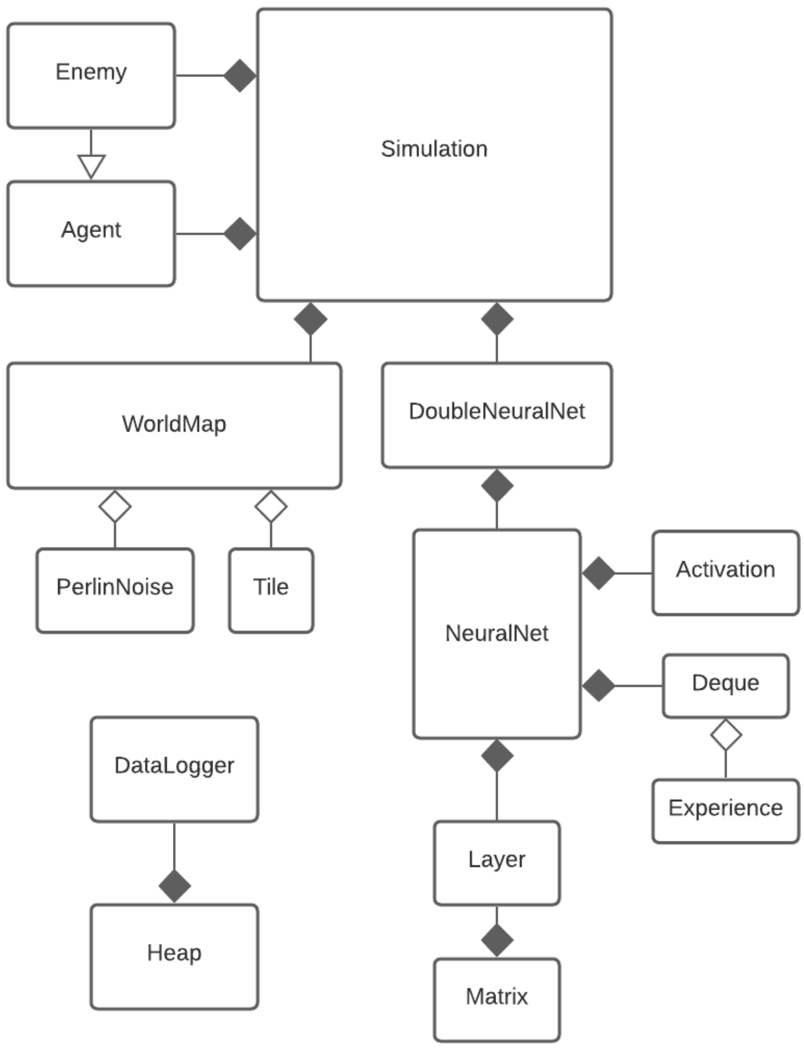
Shown below is shown a Diagram with the appropriate reward values for each Tile in the sample situation. Along with a flow chart detailing the steps it takes to calculate the reward of a given Action taken by the Agent. All reward gained during the flow process is added to a total and returned as part of the Reward Function.





2.5 Object Orientated Structure

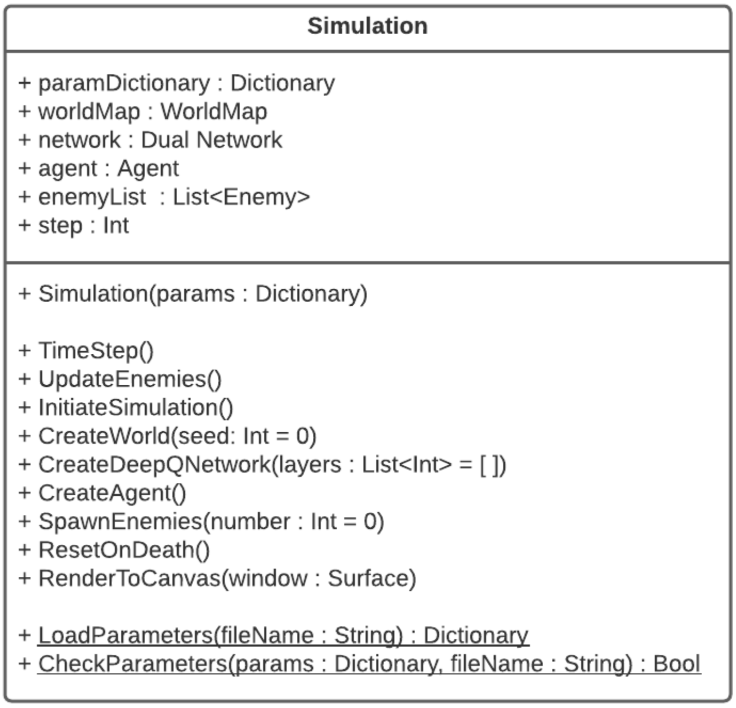
My project is formed from several classes, each having a specific role. They primarily use composition, due to being tightly linked together. Although some links are more similar to aggregation. I designed my project to primarily break down into two parts, the Neural Network and the Environment. These are split into three primary classes, which link together through the use of a Management class. This management class oversees and organises all Primary aspects of the program, storing instances of its child classes, and passing references to eachother through Methods. It also handles the Setup and Resetting for the entire project, if the Agent is killed it detects this and will signal the Environment to regenerate, create new enemies set the Agents new spawn position. Below is shown an abstraction of the Full Class Diagram, utilising only the names of the classes due to each class having too many methods to display fully. The Full Class Diagram is shown on its own page at the end of this Section.



As you can see from the Class Diagram there is alot of classes within my project, all linked together in various different ways. Below is shown a more indepth dive into each Class' Purpose, along with Explanations of their Primary Methods.

2.5.1 Simulation Class

The Simulation Class is the Management Class of the whole Program. This will be instantiated upon the launch of the program, and all user interactions with the program will be done through it. The Methods within this class are designed to be very abstracted from the core of the program. There are never any direct Matrix Operations or logic performed in this Class except for checking the parameters, and basic setup such as instantiating new Enemies *if they are enabled*. It does however call Methods which then run this logic.



The Simulation Class

2.5.2 Agent, Enemy and WorldMap Class

Both these Classes will be used by the Simulation Class to form the Environment and its interactions. The World Class will host the Procedural Generation Methods along with storing any Terrain related data. Perlin Noise and Poisson Disc Sampling are used here to generate the Height Map for the Terrain and Generate Objects. The Agent Class is used to interact with the Environment, Methods for Sampling the Tile Data and processing this into Grayscale Values for use in the Dual Neural Network. The Agent also has Reward Methods for calculating the Reward when given a certain State and Action. This is shown under the Agent Interaction and Reward Design Section.

The Enemy Class inherits from the Agent Class, implementing its own Commit Action and Spawn Position Method. This inheritance is so that any Agent Methods which might be needed in the final implementation. This fulfils objectives under **Objective 4**

WorldMap

+ mapSize : Int
+ tileWidth : Int
+ seed : Int
+ tileBorder : Int
+ tileArray : List<List<Tile>>
+ paramDictionary : Dictionary
+ renderedMap : Surface
+ renderedInteractables : Surface

+ WorldMap(seed : Int, params : Dictionary)

+ GenerateMap()
+ GenerateThreaded()
+ ThreadedChild(quadrant : List<List<Tile>>)
+ GenerateTreeArea()
+ PoissonDiscSampling(pointList : List) : List
+ PoissonCheckPoint(point : Tuple (int, int), pickedPoints : List
+ RenderMap() : Surface
+ RenderInteractables() : Surface
+ DrawMap() : Surface
+ Clamp(value : Float, low : Float, high : Float) : Float

The WorldMap Class

Perlin Noise

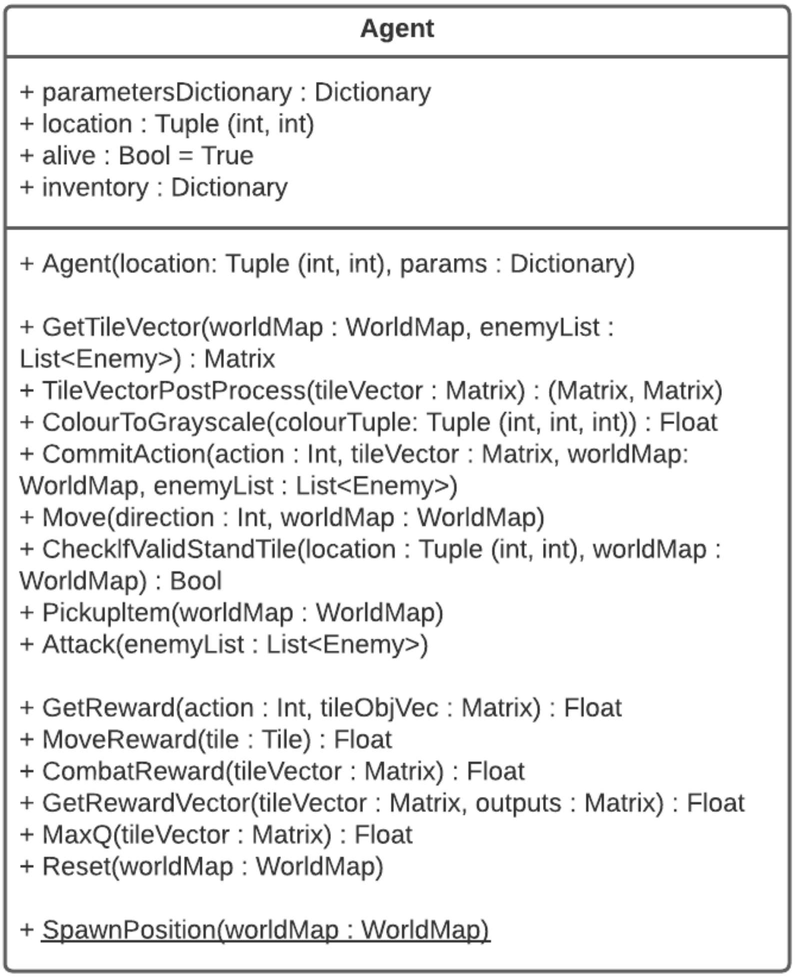
+ OctaveNoise() : Float
+ Noise() : Float
+ Gradient() : Float
+ LinearInterpolate() : Float
+ Fade() : Float

The PerlinNoise Class

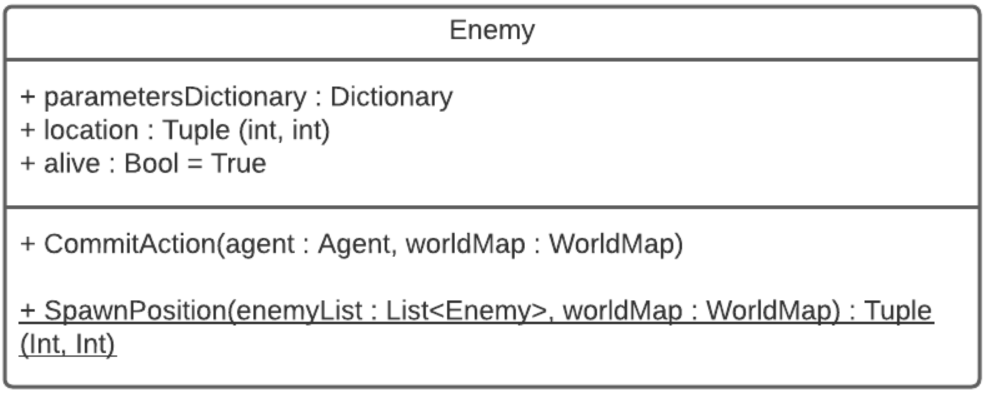
Tile

+ tileHeight : Float
+ tileType : Int
+ tileColour : Tuple (int, int, int)
+ explored : Bool = False
+ hasObject : Bool = False
+ hasEnemy: Bool = False

The Tile Class



The Agent Class



The Enemy Class

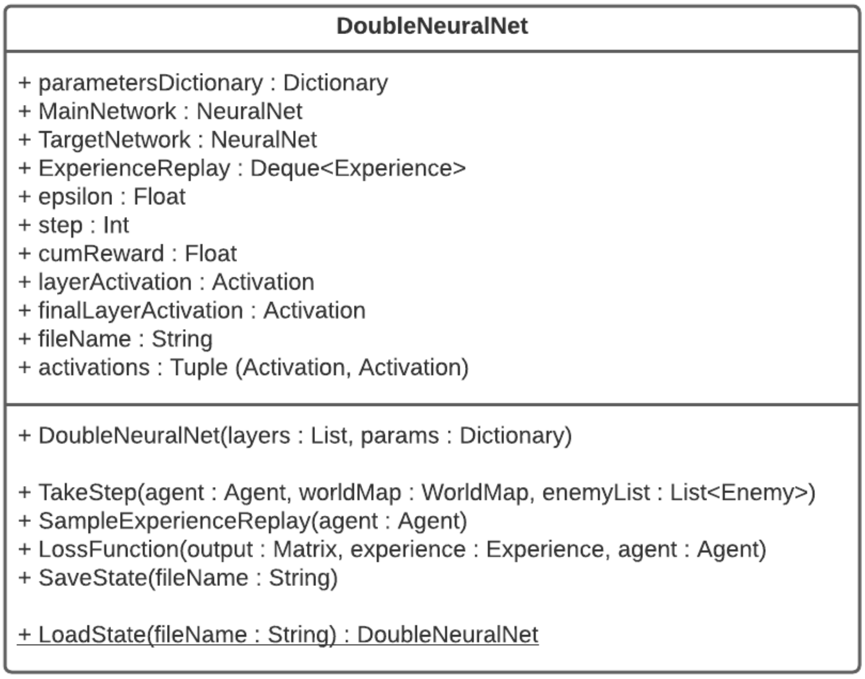
2.5.3 DoubleNeuralNet Class Family

The DoubleNeuralNet Class is instantiated as part of the setup in Simulation. This class is the parent to both the Main and Target Neural Networks, which are both instances of NeuralNet. DoubleNeuralNet hosts the Top-Level methods for invoking a Time Step within the Network. The TimeStep Method will be called from Simulation, into this method is passed references to the current WorldMap along with the current Agent. This then Forward Propagates the Main and Target Neural Networks, performs the logic for calculating the correct action utilising SoftMax and Epsilon Greedy. The expected values, and subsequently the Loss are calculated, which is then used to Back Propagate the Main Network.

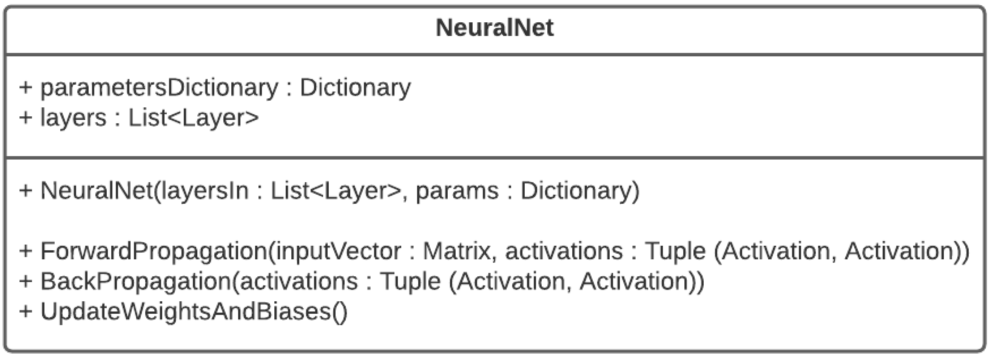
The NeuralNet Class holds a list of Layer Objects, along with intermediate Forward Prop, Back Prop and Update Methods. Upon initialisation of the NeuralNet Object, it will create the list of Layers with the specified sizes from the User. The contained Methods simply perform the Layer→Layer Logic which is needed for Forward and Back Propagation.

Each Layer Object holds Weight, Bias, Preactivation, and Activation Data. This is all utilised within the Forward and Back Propagation Process. This data is all stored in Matrices to speed up calculations.

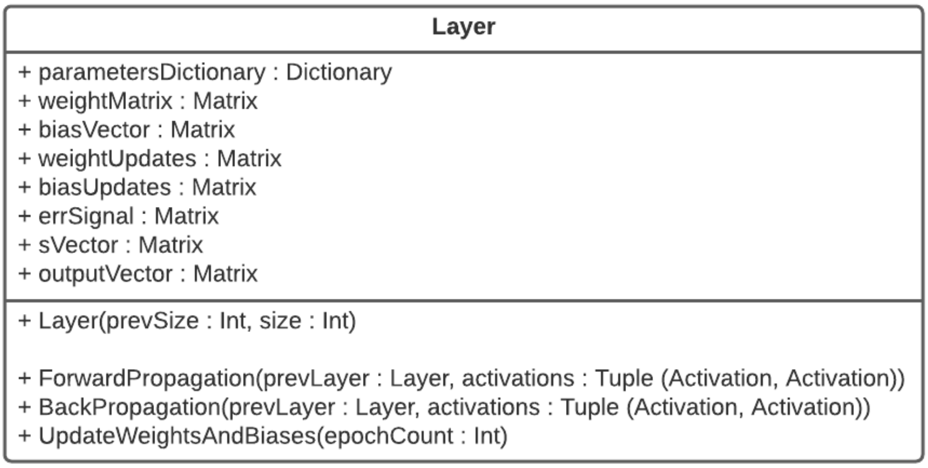
This fulfils objectives under **Objective 5**



The DoubleNeuralNet Class



The NeuralNet Class

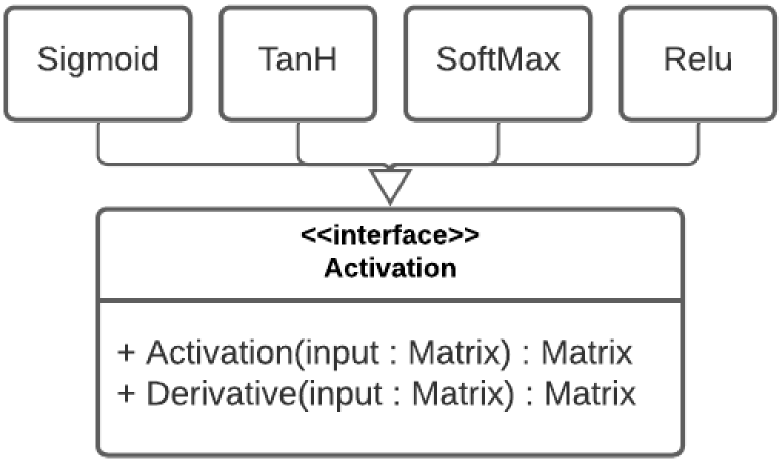


The Layer Class

2.5.4 Activation Class

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. There are two Methods which a new Activation needs to implement, Activation and Derivative. The Activations I will implement are shown in Modelling of the Problem, along with shown below in the UML Diagram.

This fulfils objectives under **Objective 6**

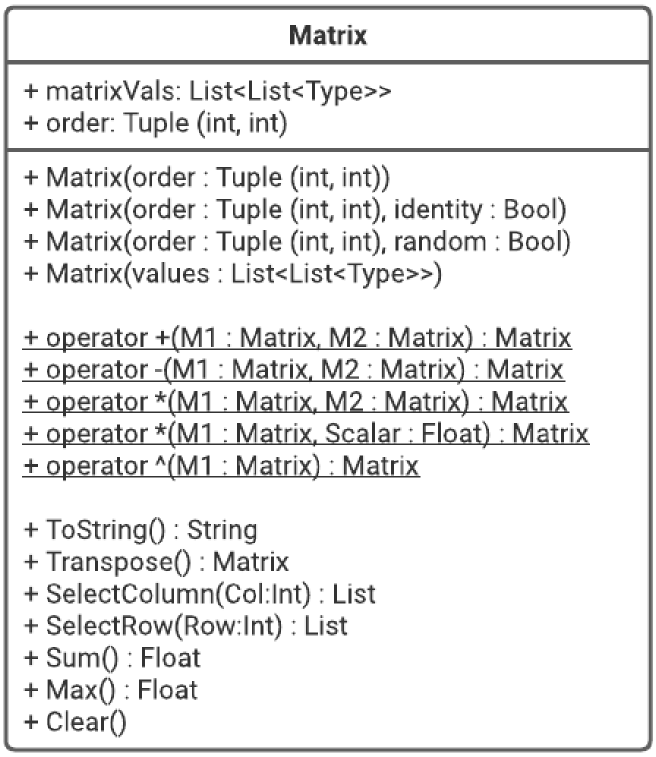


The Activation Base Class and Implemented Activations

2.5.5 Matrix Class

The Matrix Class will contain the standard Matrix Operations, along with multiple methods of creating different Matrices. It is used as an integral part of the NeuralNet Class, because it heavily relies on Matrix Operations. These Operations require efficient algorithms, which are outlined under Description of Algorithms.

This fulfils objectives under **Objective 3**

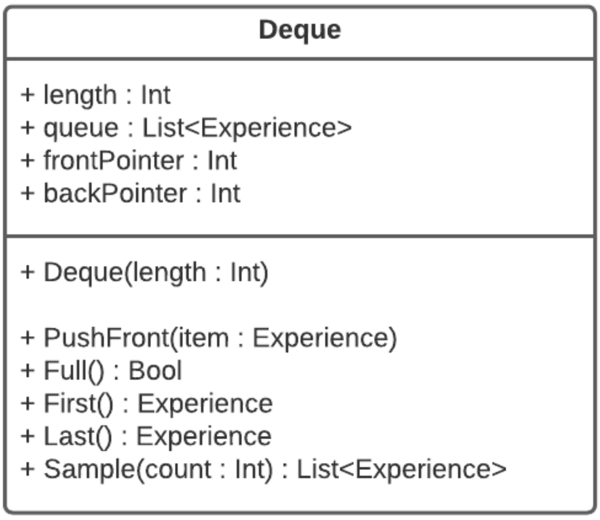


The Matrix Class

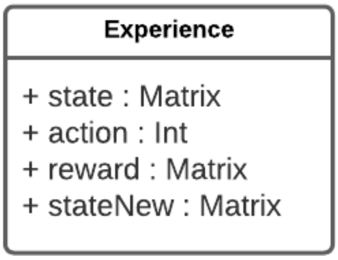
2.5.6 Deque Class and Experience Replay

The Deque Class is used as part of the Implementation of Experience Replay. Within the Deque will be stored instances of the Experience Object, which is used to store Individual State Data. The Deque will have Methods to sample states from it's contents, which will then be used within the DoubleNeuralNet Class to Back Propagate the Calculated Loss.

This fulfils objectives under **Objective 5**



The Deque Class



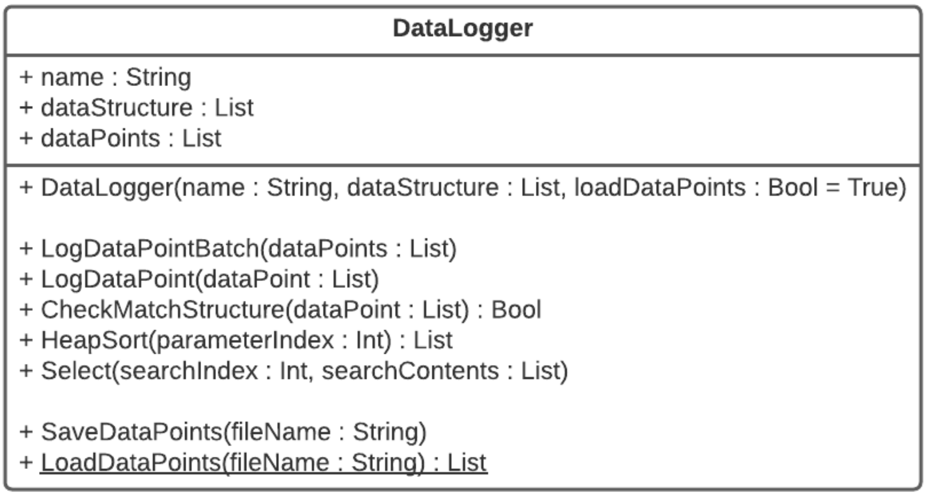
The Experience Class

2.5.7 Data Logger and Heap Class

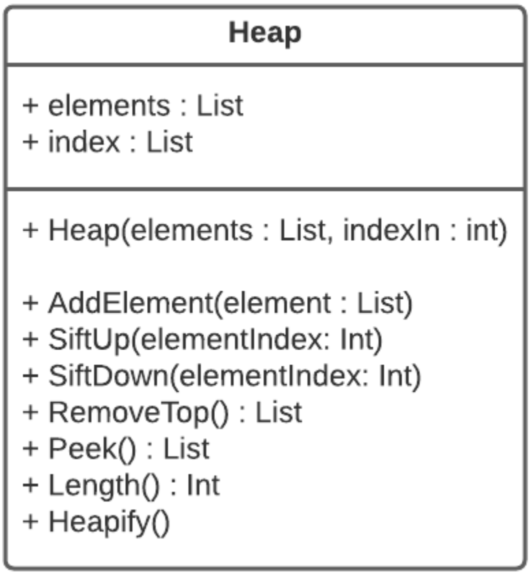
The Data Logger Class will be used to Log Data points at certain Points of the Program and then subsequently saving them to a .data File. Each Data Point added to the Logger will be checked against the Loggers Structure, which is stored as a List of Types.

These .data files can then be plotted using a Graphing Library.

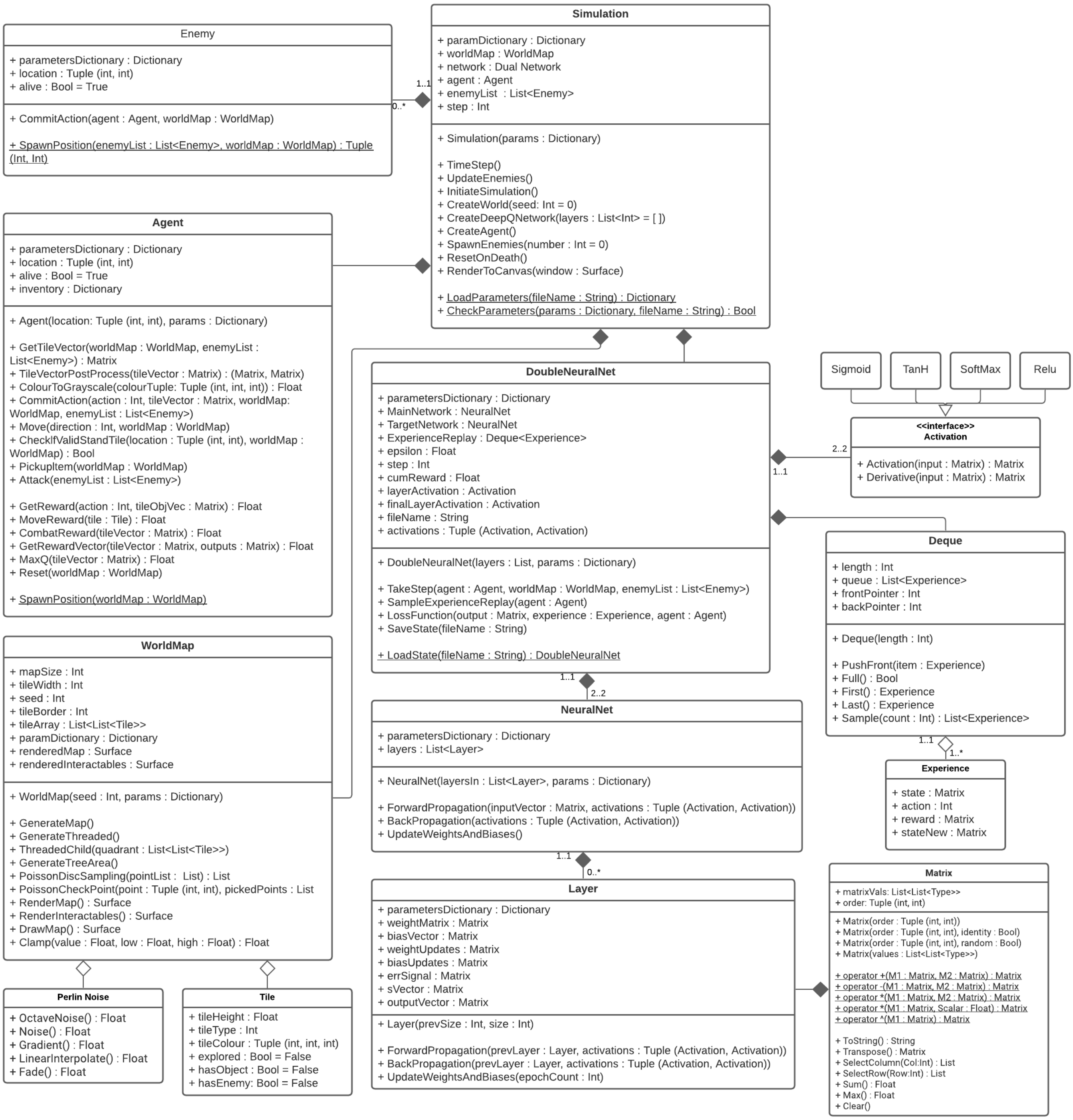
This fulfils objectives under **Objective 8**



The Deque Class



The Experience Class


Enemy
+ parametersDictionary : Dictionary
+ location : Tuple (int, int)
+ alive : Bool = True
+ CommitAction(agent : Agent, worldMap : WorldMap)
+ SpawnPosition(enemyList : List<Enemy>, worldMap : WorldMap) : Tuple (Int, Int)
1..1
0..*
Simulation
+ paramDictionary : Dictionary
+ worldMap : WorldMap
+ network : Dual Network
+ agent : Agent
+ enemyList : List<Enemy>
+ step : Int
+ Simulation(params : Dictionary)
+ TimeStep()
+ UpdateEnemies()
+ InitiateSimulation()
+ CreateWorld(seed: Int = 0)
+ CreateDeepQNetwork(layers : List<Int> = [ ])
+ CreateAgent()
+ SpawnEnemies(number : Int = 0)
+ ResetOnDeath()
+ RenderToCanvas(window : Surface)
+ LoadParameters(fileName : String) : Dictionary
+ CheckParameters(params : Dictionary, fileName : String) : Bool
Agent
+ parametersDictionary : Dictionary
+ location : Tuple (int, int)
+ alive : Bool = True
+ inventory : Dictionary
+ Agent(location: Tuple (int, int), params : Dictionary)
+ GetTileVector(worldMap : WorldMap, enemyList : List<Enemy>) : Matrix
+ TileVectorPostProcess(tileVector : Matrix) : (Matrix, Matrix)
+ ColourToGrayscale(colourTuple: Tuple (int, int, int)) : Float
+ CommitAction(action : Int, tileVector : Matrix, worldMap: WorldMap, enemyList : List<Enemy>)
+ Move(direction : Int, worldMap : WorldMap)
+ CheckIfValidStandTile(location : Tuple (int, int), worldMap : WorldMap) : Bool
+ PickupItem(worldMap : WorldMap)
+ Attack(enemyList : List<Enemy>)
+ GetReward(action : Int, tileObjVec : Matrix) : Float
+ MoveReward(tile : Tile) : Float
+ CombatReward(tileVector : Matrix) : Float
+ GetRewardVector(tileVector : Matrix, outputs : Matrix) : Float
+ MaxQ(tileVector : Matrix) : Float
+ Reset(worldMap : WorldMap)
+ SpawnPosition(worldMap : WorldMap)
DoubleNeuralNet
+ parametersDictionary : Dictionary
+ MainNetwork : NeuralNet
+ TargetNetwork : NeuralNet
+ ExperienceReplay : Deque<Experience>
+ epsilon : Float
+ step : Int
+ cumReward : Float
+ layerActivation : Activation
+ finalLayerActivation : Activation
+ fileName : String
+ activations : Tuple (Activation, Activation)
+ DoubleNeuralNet(layers : List, params : Dictionary)
+ TakeStep(agent : Agent, worldMap : WorldMap, enemyList : List<Enemy>)
+ SampleExperienceReplay(agent : Agent)
+ LossFunction(output : Matrix, experience : Experience, agent : Agent)
+ SaveState(fileName : String)
+ LoadState(fileName : String) : DoubleNeuralNet
Sigmoid
TanH
SoftMax
Relu
<<interface>>
Activation
+ Activation(input : Matrix) : Matrix
+ Derivative(input : Matrix) : Matrix
2..2
1..1
Deque
+ length : Int
+ queue : List<Experience>
+ frontPointer : Int
+ backPointer : Int
+ Deque(length : Int)
+ PushFront(item : Experience)
+ Full() : Bool
+ First() : Experience
+ Last() : Experience
+ Sample(count : Int) : List<Experience>
1..1
1..*
Experience
+ state : Matrix
+ action : Int
+ reward : Matrix
+ stateNew : Matrix
WorldMap
+ mapSize : Int
+ tileWidth : Int
+ seed : Int
+ tileBorder : Int
+ tileArray : List<List<Tile>>
+ paramDictionary : Dictionary
+ renderedMap : Surface
+ renderedInteractables : Surface
+ WorldMap(seed : Int, params : Dictionary)
+ GenerateMap()
+ GenerateThreaded()
+ ThreadedChild(quadrant : List<List<Tile>>)
+ GenerateTreeArea()
+ PoissonDiscSampling(pointList : List) : List
+ PoissonCheckPoint(point : Tuple (int, int), pickedPoints : List
+ RenderMap() : Surface
+ RenderInteractables() : Surface
+ DrawMap() : Surface
+ Clamp(value : Float, low : Float, high : Float) : Float
1..1
2..2
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()
1..1
0..*
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)
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>>)
+ operator +(M1 : Matrix, M2 : Matrix) : Matrix
+ operator -(M1 : Matrix, M2 : Matrix) : Matrix
+ operator *(M1 : Matrix, M2 : Matrix) : Matrix
+ operator *(M1 : Matrix, Scalar : Float) : Matrix
+ operator ^(M1 : Matrix) : Matrix
+ ToString() : String
+ Transpose() : Matrix
+ SelectColumn(Col:Int) : List
+ SelectRow(Row:Int) : List
+ Sum() : Float
+ Max() : Float
+ Clear()
Perlin Noise
+ OctaveNoise() : Float
+ Noise() : Float
+ Gradient() : Float
+ LinearInterpolate() : Float
+ Fade() : Float
Tile
+ tileHeight : Float
+ tileType : Int
+ tileColour : Tuple (int, int, int)
+ explored : Bool = False
+ hasObject : Bool = False
+ hasEnemy: Bool = False

2.6 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.

2.6.1 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.

```
SUBROUTINE MatrixAddition(Matrix1, Matrix2)
    TemporaryMatrix ← NEW Matrix(Matrix1.Order)
    FOR Row ← 0 TO Matrix1.Order[0]
        FOR Column ← 0 TO Matrix1.Order[1]
            TemporaryMatrix[Row, Column] ← Matrix1[Row, Column] + Matrix2[Row, Column]
        END FOR
    END FOR
    RETURN TemporaryMatrix
ENDSUBROUTINE
```

2.6.2 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.

```
SUBROUTINE MatrixSubtraction(Matrix1, Matrix2)
    TemporaryMatrix ← NEW Matrix(Matrix1.Order)
    FOR Row ← 0 TO Matrix1.Order[0]
        FOR Column ← 0 TO Matrix1.Order[1]
            TemporaryMatrix[Row, Column] ← Matrix1[Row, Column] - Matrix2[Row, Column]
        END FOR
    END FOR
    RETURN TemporaryMatrix
ENDSUBROUTINE
```

2.6.3 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 x n and Matrix B has dimensions of j x k, the resultant Matrix will have dimensions of n x 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.

```
SUBROUTINE MatrixMultiplication(Matrix1, Matrix2)
    tempMatrix ← NEW Matrix((Matrix1.Order[0], Matrix2.Order[1]))
    FOR i ← 0 TO Matrix1.Order[0]
        FOR j ← 0 TO Matrix2.Order[1]
            FOR l ← 0 TO Matrix.Order[1]
                tempMatrix[i, j] ← tempMatrix[i, j] + Matrix1[i, k] * Matrix2[k, j]
```

```
            END FOR
        END FOR
    END FOR
    RETURN tempMatrix
ENDSUBROUTINE
```

2.6.4 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.

```
SUBROUTINE MatrixScalarMultiplication(Scalar, Matrix)
    TemporaryMatrix ← NEW Matrix(Matrix.Order)
    FOR Row ← 0 TO Matrix.Order[0]
        FOR Column ← 0 TO Matrix.Order[1]
            TemporaryMatrix[Row, Column] ← Scalar * Matrix[Row, Column]
        END FOR
    END FOR
    RETURN TemporaryMatrix
ENDSUBROUTINE
```

2.6.5 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.

```
SUBROUTINE MatrixHadamardProduct(Matrix1, Matrix2)
    TemporaryMatrix ← NEW Matrix(Matrix1.Order)
    FOR Row ← 0 TO Matrix1.Order[0]
        FOR Column ← 0 TO Matrix1.Order[1]
            TemporaryMatrix[Row, Column] ← Matrix1[Row, Column] * Matrix2[Row, Column]
        END FOR
    END FOR
    RETURN TemporaryMatrix
ENDSUBROUTINE
```

2.6.6 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.

```
SUBROUTINE MatrixHadamardProduct(Matrix, Power)
    TemporaryMatrix ← CLONE Matrix
    FOR Row ← 0 TO Power - 1
        TemporaryMatrix ← TemporaryMatrix * Matrix
    END FOR
    RETURN TemporaryMatrix
ENDSUBROUTINE
```

2.6.7 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 x n Matrix will turn into an n x m Matrix.

```
SUBROUTINE MatrixTranspose(Matrix)
    TemporaryMatrix ← NEW Matrix(Matrix.Order)
    FOR Row ← 0 TO Matrix.Order[0]
        FOR Column ← 0 TO Matrix.Order[1]
            TemporaryMatrix[Row, Column] ← Matrix[Column, Row]
        END FOR
    END FOR
    RETURN temporaryMatrix
ENDSUBROUTINE
```

2.6.8 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.

```
SUBROUTINE Softmax(Input)
    OutVector ← NEW Matrix(Input.Order)
    ExpSum ← 0
    FOR Row ← 0 TO Input.Order[0]
        ExpSum ← ExpSum + Math.exp(Input[Row, 0])
    END FOR
    FOR Row ← 0 TO Input.Order[0]
        OutVector[Row] ← Input[Row, 0] / ExpSum
    END FOR
    RETURN OutVector
ENDSUBROUTINE
```

2.6.9 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.

```
SUBROUTINE Forward Propagation(PrevLayer, Activations, FinalLayer)
    WeightValueProduct ← This.WeightMatrix * PrevLayer.OutputVector
    This.SVector ← WeightValueProduct + This.BiasVector
    IF NOT FinalLayer
        This.OutputLayer ← Activations[0].Activation(SVector)
    ELSE
        This.OutputLayer ← Activations[1].Activation(SVector)
    END IF
ENDSUBROUTINE
```

2.6.10 Half Square Difference

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

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

2.6.11 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 neccesary 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 Inital Action and the Value of the Potential Payoff after taking that Initial Action.

2.6.12 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 derivates 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.

```
SUBROUTINE BackPropagation(PreviousLayer, LearningRate, Activation)
    WeightTranspose ← PreviousLayer.WeightMatrix.Transpose()
    DeltaWeightProduct ← WeightTranspose * PreviousLayer.ErrorSignal
    This.ErrorSignal ← DeltaWeightProduct * Activation.Derivative(This.PreActivations)

    WeightDerivatives ← This.ErrorSignal * This.Activations.Transpose()
    BiasDerivatives ← This.ErrorSignal

    This.WeightUpdates ← This.WeightUpdates + (WeightDerivatives * LearningRate)
    This.BiasUpdates ← This.BiasUpdates + (BiasDerivatives * LearningRate)
ENDSUBROUTINE
```

2.6.13 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.

```
SUBROUTINE ExperienceReplay(SampleSize, Agent)
    Samples ← NEW List()
    FOR i ← 0 TO SampleSize
        Samples.Add(Buffer.RandomSample())
    END FOR

```

```
    FOR Sample IN Samples
        PostProcessedSurround ← Agent.TileVectorPostProcess(sample.state)

        NetInput ← PostProcessedSurround[1]

        This.MainNetwork.ForwardPropagation(NetInput, This.Activation)

        Output ← This.MainNetwork.Layers[-1].Activations

        ExpectedValues ← This.ExpectedValue(Output, Sample, Agent)

        Cost ← This.HalfSquareDiff(Output, ExpectedValues)

        Preactivations ← This.MainNetwork.Layers[-1].Preactivations
        PreactivationsDerivative ← This.Activation.Derivative(Preactivations)
        This.MainNetwork.Layers.ErrSignal ← Cost * PreactivationsDerivative

        This.MainNetwork.BackPropagation(This.Activation)
    END FOR
ENDSUBROUTINE
```

2.6.14 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.

```
SUBROUTINE GetTileVector(WorldMap)
    Offset ← LoadFromParameters("DQLOffset")
    SideLength ← 2 * Offset + 1
    TileVector ← NEW Matrix((Math.pow(sideLength, 2), 1))
    Num ← 0
    FOR i ← Agent.Pos[1] - Offset TO Agent.Pos[1] + Offset + 1
        FOR j ← Agent.Pos[0] - Offset TO Agent.Pos[1] + Offset + 1
            TileVector[Num, 0] ← WorldMap[j, i]
            Num ← Num + 1
        END FOR
    END FOR
    RETURN TileVector
ENDSUBROUTINE
```

2.6.15 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.

```
SUBROUTINE RGBToGrayscale(RGBVal)
    GrayscaleValue ← 0
    GrayscaleValue ← GrayscaleValue + (0.299 * RGBVal[0])
    GrayscaleValue ← GrayscaleValue + (0.587 * RGBVal[1])
    GrayscaleValue ← GrayscaleValue + (0.114 * RGBVal[2])
    RETURN GrayscaleValue / 255
ENDSUBROUTINE
```

2.6.16 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.

```
SUBROUTINE GetTileVector(TileVector)
    ProcessedVector ← NEW Matrix(TileVector.Order)
    FOR Row ← 0 TO TileVector.Order[0]
        ProcessedVector[Row, 0] ← RGBToGrayscale(TileVector[Row, 0].RGBValue)
    END FOR
    RETURN ProcessedVector
ENDSUBROUTINE
```

2.6.17 Agent Spawn Position

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

```
SUBROUTINE AgentSpawnPosition(WorldMap)
    SpawnList ← NEW List()
    MapSize ← LoadFromParameters("MapSize")
    FOR y ← 0 TO MapSize
        FOR x ← 0 TO MapSize
            IF WorldMap[x, y].TileType == 2
                SpawnList.Add([x, y])
            END IF
        END FOR
    END FOR
    SpawnList.Shuffle()
    RETURN SpawnList[0]
ENDSUBROUTINE
```

2.6.18 Enemy Spawn Position

This algorithm will create a list of spawnable tiles for which Enemies can spawn on, then select tiles randomnly, if they dont already contain an enemy or the agent it will create an Enemy Object with that position. It will do this n ammount of times where n is the limit to how many enemies can spawn.

```
SUBROUTINE EnemySpawnPosition(WorldMap, EnemyList)
    SpawnList ← NEW List()
    EnemyLocationList ← NEW List()
    MapSize ← LoadFromParameters("MapSize")
    FOR y ← 0 TO MapSize
        FOR x ← 0 TO MapSize
            IF WorldMap[x, y].TileType == 2
                SpawnList.Add([x, y])
            END IF
        END FOR
    END FOR
    SpawnList.Shuffle()
    IF SpawnList[0] IN EnemyLocationList
        RETURN NONE
    ELSE
        RETURN SpawnList[0]
    END IF
```

```
    RETURN SpawnList[0]
ENDSUBROUTINE
```

2.6.19 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.

```
SUBROUTINE EnemyMove(Agent, WorldMap)
    XDifference ← Agent.Pos[0] - This.Pos[0]
    YDifference ← Agent.Pos[1] - This.Pos[0]

    IF XDifference == 0 AND YDifference == 0
        Agent.Alive = False
        RETURN
    END IF

    IF abs(XDifference) > abs(YDifference)
        IF XDifference > 0
            This.Pos[0] ← This.Pos[0] + 1
        ELSE
            This.Pos[0] ← This.Pos[0] - 1
        END IF
    ELSE IF abs(XDifference) < abs(YDifference)
        IF YDifference > 0
            This.Pos[1] ← This.Pos[1] + 1
        ELSE
            This.Pos[1] ← This.Pos[1] - 1
        END IF
    END IF
ENDSUBROUTINE
```

2.6.20 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.

```
SUBROUTINE PoissonDiscSampling(PointList)
    KVal ← LoadFromParameters("PoissonKVal")
    MapSize ← LoadFromParameters("MapSize")
    PickedPoints ← NEW Grid(MapSize, MapSize)
    SampleNum ← LoadFromParameters("MapSize")
    WHILE SampleNum <= KVal
        Sample ← PointList[RandomInt(0, PointList.Length - 1)]
        Result ← CheckPointDistance(Sample, PickedPoints)
        IF Result == True
            PickedPoints[Sample[0], Sample[1]] ← True
            SampleNum ← 0
```

```
            CONTINUE
        ELSE
            SampleNum ← SampleNum + 1
            CONTINUE
        END IF
    END WHILE
    RETURN PickedPoints
ENDSUBROUTINE
```

2.6.21 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.

```
PermTable ← [1 → 255].Shuffle() * 2

SUBROUTINE PerlinNoise(X, Y)
    XFloor ← Math.floor(X)
    YFloor ← Math.floor(Y)

    G1 ← PermTable[PermTable[XFloor] + YFloor]
    G2 ← PermTable[PermTable[XFloor + 1] + YFloor]
    G3 ← PermTable[PermTable[XFloor] + YFloor + 1]
    G4 ← PermTable[PermTable[XFloor + 1] + YFloor + 1]

    XExact ← X - XFloor
    YExact ← Y - YFloor

    D1 ← Grad(G1, XFloor, YFloor)
    D2 ← Grad(G2, XFloor - 1, YFloor)
    D3 ← Grad(G3, XFloor, YFloor - 1)
    D4 ← Grad(G4, XFloor - 1, YFloor - 1)

    U ← Fade(XFloor)
    V ← Fade(YFloor)

    XInterpolated ← Lerp(U, D1, D2)
    YInterpolated ← Lerp(U, D3, D4)

    RETURN Lerp(V, XInterpolated, YInterpolated)
ENDSUBROUTINE

SUBROUTINE Grad(Hash, X, Y)
    Temp ← Hash BITWISEAND 3
    IF Temp == 0
        RETURN X + Y
    ELSE IF Temp == 1
        RETURN -X + Y
    ELSE IF Temp == 2
        RETURN X - Y
    ELSE IF Temp == 3
        RETURN -X - Y
    ELSE
```

```
        RETURN 0
    END IF
ENDSUBROUTINE

SUBROUTINE Lerp(Ammount, Left, Right)
    RETURN ((1 - Ammount) * Left + Ammount * Right)
ENDSUBROUTINE

SUBROUTINE Fade(T)
    RETURN T * T * T * (T * (T * 6 - 15) + 10)
ENDSUBROUTINE
```

2.6.22 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.

```
SUBROUTINE OctaveNoise(X, Y, Octaves, Persistence)
    Total ← 0
    Frequency ← 1
    Amplitude ← 1
    MaxValue ← 0

    FOR i ← 0 TO Octaves
        Total ← Total + (PerlinNoise(X * Frequency, Y * Frequency) * Amplitude

        MaxValue ← MaxValue + Amplitude

        Amplitude ← Amplitude * Persistence
        Frequency ← Frequency * 2
    END FOR

    RETURN Total / MaxValue
ENDSUBROUTINE
```

2.6.23 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 $\lfloor \mathbf{(N-1)/2} \rfloor$ 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 relys on the fact that Children of an Index are located at $\mathbf{2i + 1}$ and $\mathbf{2i + 2}$.

```
SUBROUTINE Heapify()
    FOR i ← ⌊(HeapList.Length-1)/2⌋ TO 0 STEP -1
        SiftDown(i)
    END FOR
ENDSUBROUTINE

SUBROUTINE SiftDown(RootIndex)
    IsHeap ← FALSE
    End ← HeapList.Length - 1

    WHILE (2 * RootIndex) + 1 <= End
```

```
        ChildIndex = (RootIndex * 2) + 1
        IF ChildIndex <= End AND HeapList[ChildIndex] < HeapList[ChildIndex + 1]
            ChildIndex ← ChildIndex + 1
        END IF
        IF HeapList[RootIndex] < HeapList[ChildIndex]
            TempSwap ← HeapList[ChildIndex]
            HeapList[ChildIndex] ← HeapList[RootIndex]
            HeapList[RootIndex] ← TempSwap
        ELSE
            BREAK
        END IF
ENDSUBROUTINE
```

2.6.24 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.

```
SUBROUTINE RemoveTop()
    TempSwap ← HeapList[-1]
    HeapList[-1] ← HeapList[0]
    HeapList[0] ← TempSwap
    ReturnItem ← HeapList.Pop()

    Heapify()

    RETURN ReturnItem
ENDSUBROUTINE
```

2.6.25 Heap Sort

The Heap Sort algorithm relys on the prior two algorithms to fully order a list in Worst and Best case $O(nlog(n))$ 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.

```
SUBROUTINE HeapSort()
    SortedList ← NEW List()
    Heap ← NEW Heap(DataPoints)

    WHILE Heap.Size() - 1 >= 0
        SortedList.Append(Heap.RemoveTop)
    END FOR

    RETURN SortedList
ENDSUBROUTINE
```

2.7 Description of Data Structures

2.7.1 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.

2.7.2 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.

2.7.3 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 attatched 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.

2.7.4 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.

2.7.5 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 $O(nlog(n))$ Sorting Algorithm. The highest priority element is always stored at the Root, with the tree of the structure being considered "Partialy Ordered". Heaps can be stored in an Array, with the Root element at Index 0. Children of an Index are located at $\mathbf{2i + 1}$ and $\mathbf{2i + 2}$. The Parent of an Index is located at $\lfloor \mathbf{(i\text{-}1)/2} \rfloor$.

2.8 File Structure

2.8.1 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 cuttoff values for Water.
Coast	Float	0 - 1	The cuttoff values for Coast.
Grass	Float	0 - 1	The cuttoff values for Grass.
Mountain	Float	0 - 1	The cuttoff values for Mountains.
TreeType	String	-	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	Wether Experience Replay is Enabled or Disabled.
ERBuffer	Int	1k - 10k	The size of the Experience Replay Buffer.
ERSampleRate	Int	1 - 100	The ammount of steps between each Experience Replay sample.
ERSampleSize	Int	10 - 1000	The ammount 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 ammount 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

2.8.2 .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.

2.8.3 .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.

2.9 Integrity and Exception Handling

2.9.1 Ranges

As mentioned above under File Structure, I have implemented a Range Json File, in order to specify the ranges of each parameter the User can input. This is used to help the user avoid breaking the simulation by creating unintended results. Things such as creating a very large Neural Network would take up alot of memory, possibly causing Memory Allocation issues with the Users system. There is no need to implement Exceptions for most sections of the Project due to the Nature of the system being Dynamically created by the users Json Input. Which is of course sanitised by the Range File.

2.9.2 Data Logger Structure Matching

As part of my Data Logger I will implement a system to check if a New Data Point Matches the structure of the Data Logger. This structure will be stored as a List of Types, such as [[Int, Float], String, Int], where a nested list represents a multi-typing scenario. This multi-typing scenario would be cases like [Int, Float] where they are interchangeable in some cases. If a New Data Point does not match the structure of the Data Logger it will throw an appropriate error.

2.9.3 Matrix Exceptions

As part of my Matrix Class I made a series of appropriate Exceptions to help while developing and implementing any Matrix related Logic. These Exceptions are thrown when any misintended methods or operations are performed. Such as Multiplying incompatably Matrices, or utilising a non-existant initialisation case. Below is shown a Table of Exceptions I have made:

Exception Identifier	Exception Message
NoMatchingInitCase	No Matching Init case for given parameters
UnableToCreateIdentityMat	Unable to create identity Matrix from given arguments
NotOfTypeVector	Given list of Vectors contains a Matrix
VectorsNotOfSameLength	All Vectors must be the same height
NoMatchingMultiplycase	No matching multiply case found
NoMatchingAdditionCase	No matching addition case found
NoMatchingSubtractionCase	No matching addition case found
NoMatchingPowerCase	No matching power case was found
MismatchOrders	Orders of Matrices do not match
SumOfMatrixReqNumericalVals	The sum of a Matrix requires Numerical values
ColumnOutOfRange	Specified Column out of range of Matrix
ColumnMustBeInteger	Specified Column must be of type Integer
RowOutOfRange	Specified Row out of range of Matrix
RowMustBeInteger	Specified Row must be of type Integer

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 $float/int$ 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 Calulation	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 $float/int$	Pass	2.19
20	Randomised Matrix Constructor Tests	Generator Constructor Parameters randomnly 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	Run program as normal	Actions are predicted by the Network	Pass	3.3
4	Loss Function	Run program as normal	Loss is calculated	Pass	3.4
5	Back Propagation Test	Run program as normal	Calculated Loss is Back Propagated through the Network	Pass	3.5
6	Deque Push Front	A value to push to the Deque	Item is pushed to front of Deque	Pass	3.6
7	Deque First/Last	Call the .First() or .Last() Method for a Deque Object	Returns item at Front/Last index of Deque	Pass	3.7
8	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	3.8
9	Experience Replay Sampling	Run program as normal	Back Propagation is performed on the sampled Deque Items	Pass	3.9
10	Activation Outputs Unit Test	Input Value Vector to the Activation Function	Returns a Vector of values, where the Activation has been applied to them	Pass	3.10
11	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	Pass	3.11

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 .GetTileVector(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 .TileVectorPostProcess(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	Program running as normal	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,
    "DQLEpisonRegression": 0.99998,
    "DQLLearningRate": 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, 'PersistenceT
rees': 0.95, 'PoissonKVal': 20, 'TreeSeedOffset': 1000, 'TreeHeight': 0.15, 'InteractableTileBorder': 0, 'TreeBeachOffset': 0.05, 'Graysc
ale': 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, 'ERB
uffer': 1000, 'ERSampleRate': 100, 'ERSampleSize': 10, 'DeepQLearningLayers': [49, 64, 32, 16, 7], 'DQLEpoch': 100, 'DQLearningMaxSteps':
 10000, 'DQLOffset': 3, 'DQLEpsilon': 0.5, 'DQLEpisonRegression': 0.99998, 'DQLLearningRate': 0.75, 'DQLGamma': 0.8}
```

Evidence 1.2

Console Output when parameters are within specified ranges

```
Parameters within Specified Ranges
```

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

```
Parameters > Range.param
{
    "StepDelay": [0,null],

    "WorldSize": [8,1024],
    "TileWidth": [1,8],
    "TileBorder": [0,3],

    "OctavesTerrain": [0,20],
    "PersistenceTerrain": [0,1],
    "WorldScale": [0.1,null],

    "OctavesTrees": [0,20],
    "PersistenceTrees": [0,1],
    "PoissonRVal": [0,null],
    "PoissonKVal": [0,null],
    "TreeHeight": [0,1],
    "InteractableTileBorder": [0,10],
    "TreeBeachOffset": [0,1],

    "Grayscale": [0,1],
    "Water": [0,1],
    "Coast": [0,1],
    "Grass": [0,1],
    "Mountain": [0,1],

    "StartEnemyCount": [0, 100],

    "TargetReplaceRate": [5,300],
    "ERBuffer": [1000, 10000],
    "ERSampleRate": [1,100],
    "ERSampleSize": [10, 1000],

    "DQLearningMaxSteps": [0,null],
    "DQLOffset": [0,20],
    "DQLEpsilon": [0,1],
    "DQLEpisonRegression": [0,1],
    "DQLLearningRate": [0,1],
    "DQLGamma": [0,1]
}
```

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
```

Evidence 1.4

The given out of range parameter - exceeding

```
"TreeBeachOffset": 1.2,
```

The specified range it should be within

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

The Exception thrown when the program is run

```
Exception: 'TreeBeachOffset' of value 1.2, has exceeded the range: 0-1
```

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

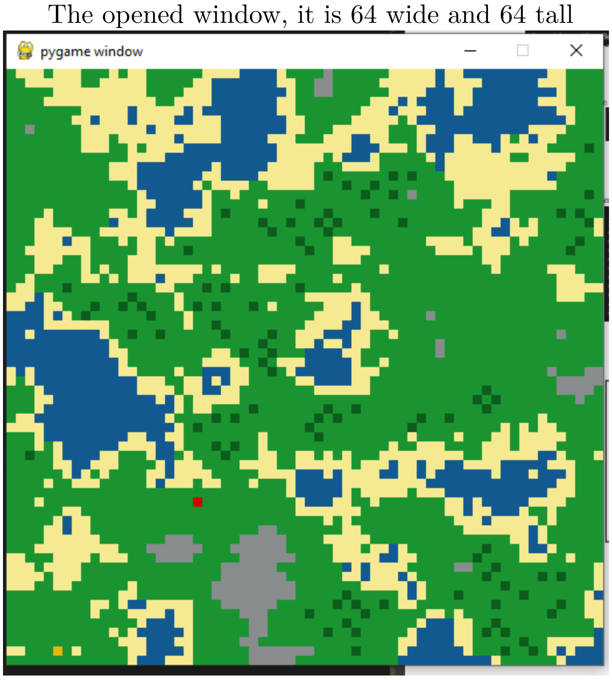
```
Load weights (Y/N): Y
State file name: DQNetwork
Created New World, Seed: 765802
Created New World, Seed: 274263
Created New World, Seed: 142187
Created New World, Seed: 613313
Created New World, Seed: 961492
Created New World, Seed: 493768
Created New World, Seed: 551641
Created New World, Seed: 133180
400 2.049999999999966 0.49601591773672193
Created New World, Seed: 310069
```

Evidence 1.6

The width/height of the window

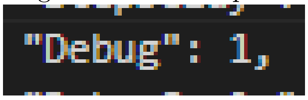
```
"WorldSize": 64,
```

The opened window, it is 64 wide and 64 tall

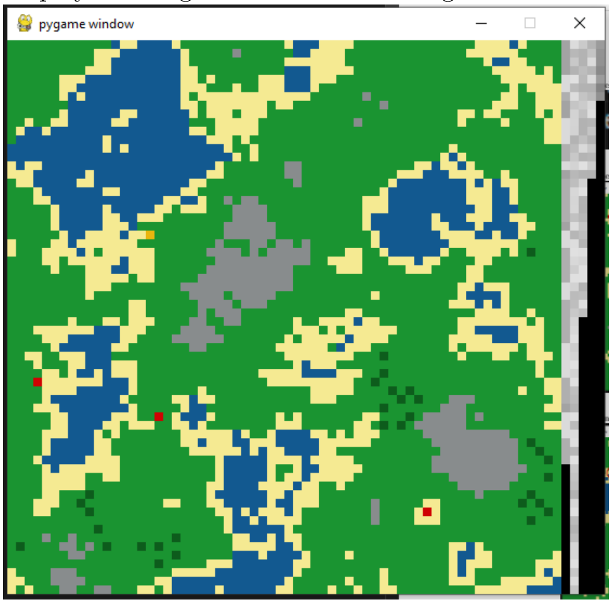


Evidence 1.7

Debug being set to 1 in the parameters file

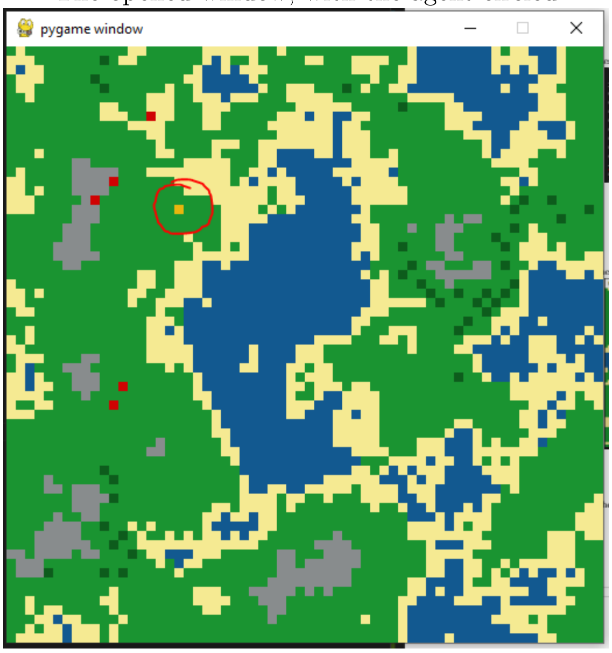


The displayed debug information to the right of the Window



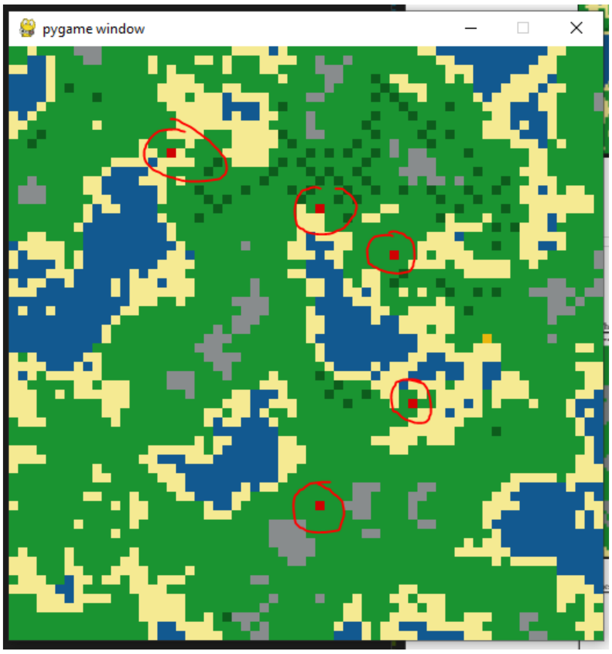
Evidence 1.8

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
```

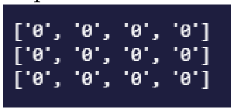
3.2.2 Matrix Implementation Tests

Evidence 2.1

Creating a Matrix with a Tuple

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

The output of the above code:



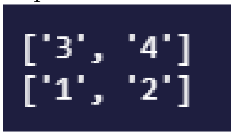
```
['0', '0', '0', '0']
['0', '0', '0', '0']
['0', '0', '0', '0']
```

Evidence 2.2

Creating a Matrix with a 2d List

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

The output of the above code:



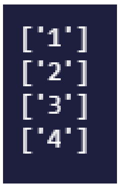
```
['3', '4']
['1', '2']
```

Evidence 2.3

Creating a Matrix with a 1d List

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

The output of the above code:



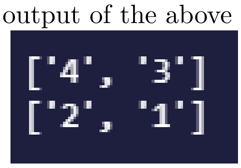
```
['1']
['2']
['3']
['4']
```

Evidence 2.4

Printing a Matrix to the console

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

The output of the above code:



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

Evidence 2.5

Creating a Randomised Matrix

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

The output of the above code:

```
['-0.20778786420611217', '-0.13982601523332772']
['0.19471852312767213', '-0.21125677633285878']
```

Evidence 2.6

Creating an Identity Matrix

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

The output of the above code:

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

Evidence 2.7

Matrix Addition Calculation

```
values = [[4, 3],
        [2, 1]]
matrix = Matrix(values)
values2 = [[3, 4],
    [1, 2]]
matrix2 = Matrix(values2)

result = matrix + matrix2
print(result)
```

The output of the above code:

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

Evidence 2.8

Matrix Subtraction Calculation

```
values = [[4, 3],
        [2, 1]]
matrix = Matrix(values)
values2 = [[3, 4],
    [1, 2]]
matrix2 = Matrix(values2)

result = matrix - matrix2
print(result)
```

The output of the above code:

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

Evidence 2.9

Matrix Multiplication Calculation

```
values = [[4, 3],
        [2, 1]]
matrix = Matrix(values)
values2 = [[3, 4],
        [1, 2]]
matrix2 = Matrix(values2)

result = matrix * matrix2
print(result)
```

The output of the above code:

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

Evidence 2.10

Matrix Scalar Multiplication Calculation

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

result = matrix * 3
print(result)
```

The output of the above code:

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

Evidence 2.11

Vector Hadamard Product Calculation

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

values = [4, 3, 2, 1]
vector2 = Matrix(values)

result = vector * vector2
print(result)
```

The output of the above code:

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

Evidence 2.12

Matrix Power Calculation

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

result = matrix ** 5
print(result)
```

The output of the above code:

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

Evidence 2.13

Matrix Transpose Calculation

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

result = matrix.Transpose()
print(result)
```

The output of the above code:

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

Evidence 2.14

Matrix Select Column

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

result = matrix.SelectColumn(1)
print(result)
```

The output of the above code:

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

Evidence 2.15

Matrix Select Row

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

result = matrix.SelectRow(0)
print(result)
```

The output of the above code:

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

Evidence 2.16

Vector Max

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

result = vector.MaxInVector()
print(result)
```

The output of the above code:

```
(6, 2)
```

Evidence 2.17

Matrix Clear

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

matrix.Clear()
print(matrix)
```

The output of the above code:

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

Evidence 2.18

Matrix Combine Vectors

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

values = [4, 3, 2, 1]
vector2 = Matrix(values)

vectorList = [vector, vector2]

result = Matrix.CombineVectorsHor(vectorList)
print(result)
```

The output of the above code:

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

Evidence 2.19

Matrix Sum

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

result = matrix.Sum()
print(result)
```

The output of the above code:

```
21
```

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 Q 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

Actions being printed to console, predicted by the Network

```
Action:  5
300 -2.7700000000000053 0.4970089522060288 2.5390241146087646
Action:  2
Action:  5
Action:  4
Action:  1
Action:  0
Action:  3
Action:  0
Action:  2
Action:  2
Action:  3
```

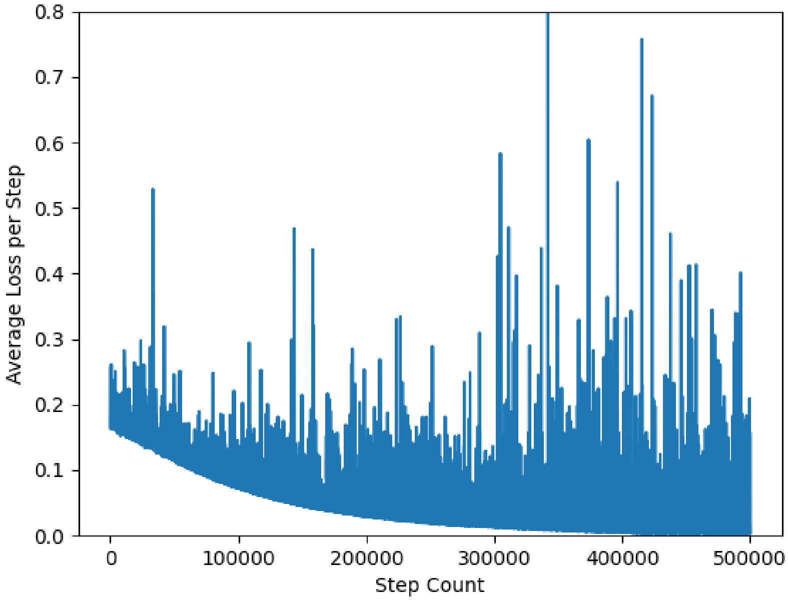
Evidence 3.4

Loss Vector printed to console

```
Loss Vector:
['2.048e-09']
['5.248800000000002e-08']
['0.0035302378626929417']
['0.003955464751067772']
['0.014010561707774592']
['3.5819647999999996e-05']
```

Evidence 3.5

Average Loss Per Step negatively declines meaning the Back Propagation is Functional



Evidence 3.6

Pushing items to the front of the Double Ended Queue

```python
deque = Deque(10)
deque.PushFront(3)
print("Added 3:", deque.queue)
deque.PushFront(-5)
print("Added -1:", deque.queue)
deque.PushFront(9)
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 3.7

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

```python
deque = Deque(4)
deque.PushFront(3)
```

```
deque.PushFront(-5)
deque.PushFront(9)
deque.PushFront(4)
deque.PushFront(-4)

print("First:", deque.First())
print("Last:", deque.Last())
print("Queue:", deque.queue)
```

The output of the above code:

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

Evidence 3.8

Create a Double Ended Queue and Sample items from the Queue

```
deque = Deque(4)
deque.PushFront(3)
deque.PushFront(-5)
deque.PushFront(9)
deque.PushFront(4)
deque.PushFront(-4)

print("Sample 1:", deque.Sample(2))
print("Sample 2:", deque.Sample(2))
print(deque.queue)
```

The output of the above code:

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

Evidence 3.9

Experience Replay buffer being sampled every 100 Steps

```
Created New World, Seed: 676738
Sampling Experience Replay
1003000 -12130.510000036405 9.703660123778806e-10 10.715813636779785
Created New World, Seed: 284115
Sampling Experience Replay
1003100 -12131.030000036402 9.684272004231704e-10 11.134362936019897
Created New World, Seed: 414746
```

The parameters in Parameters file

```
"EREnabled": 1,
"ERBuffer": 1000,
"ERSampleRate": 100,
"ERSampleSize": 10,
```

Evidence 3.10

Testing the Implemented Activations with specific values

```
outputVector = Matrix([-1, 0, 1])

```

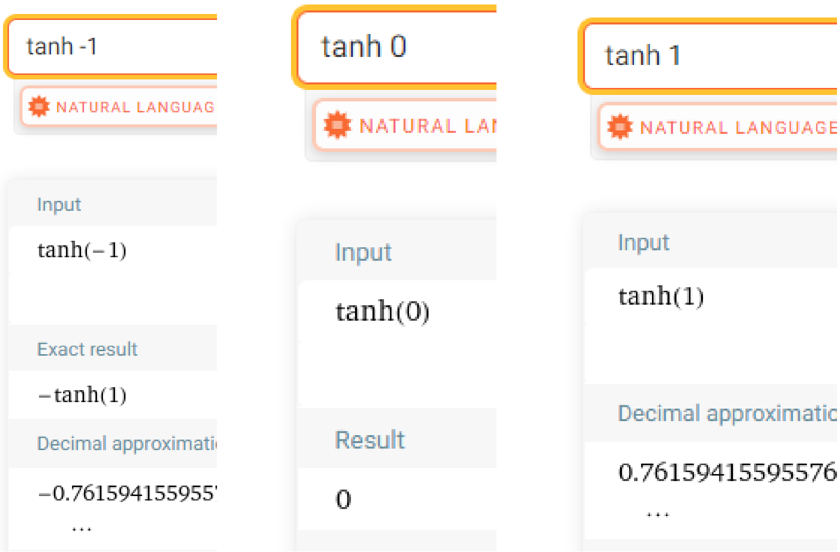
```python
Sigmoid = Sigmoid()
print("Sigmoid:")
print(Sigmoid.Activation(outputVector))

TanH = TanH()
print("TanH:")
print(TanH.Activation(outputVector))

ReLu = ReLu()
print("ReLu:")
print(ReLu.Activation(outputVector))

LeakyReLu = LeakyReLu()
print("LeakyReLu:")
print(LeakyReLu.Activation(outputVector))
```

The output of the above code:

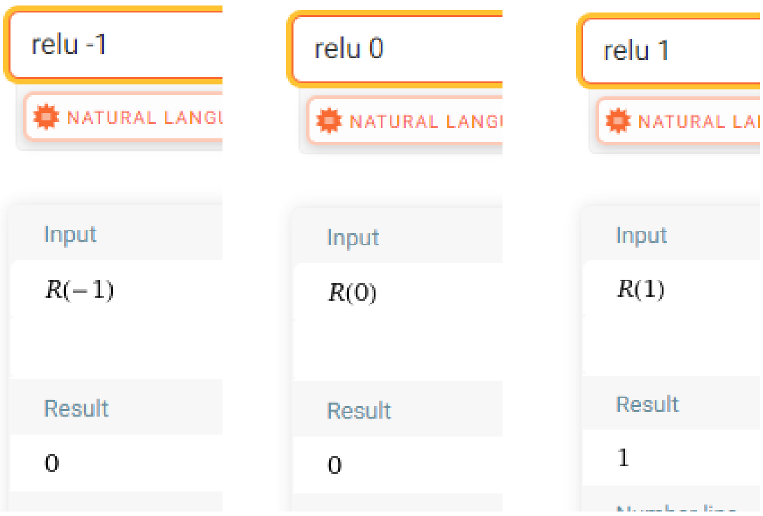
```
Sigmoid:
['0.2689414213699951']
['0.5']
['0.7310585786300049']

TanH:
['-0.7615941559557649']
['0.0']
['0.7615941559557649']

ReLu:
['0']
['0']
['1']

LeakyReLu:
['-0.1']
['0.0']
['1']
```

The WolframAlpha Sigmoid Results:

The WolframAlpha TanH Results:



The WolframAlpha ReLu Results:



The Leaky ReLu Results (Not supported by WolframAlpha):

$$
\begin{aligned}
\text{LeakyReLu}(x) &= \begin{cases} x < 0 & 0.1 \times x \\ x >= 0 & x \end{cases} \\
\text{LeakyReLu}(-1) &= -1 \times 0.1 = -0.1 \\
\text{LeakyReLu}(0) &= 1(0) = 0 \\
\text{LeakyReLu}(1) &= 1(1) = 1
\end{aligned}
$$

Evidence 3.11

Testing the Implemented Activation Derivatives with specific values

```
outputVector = Matrix([-1, 0, 1])

Sigmoid = Sigmoid()
print("Sigmoid Derivative:")
print(Sigmoid.Derivative(outputVector))

TanH = TanH()
```

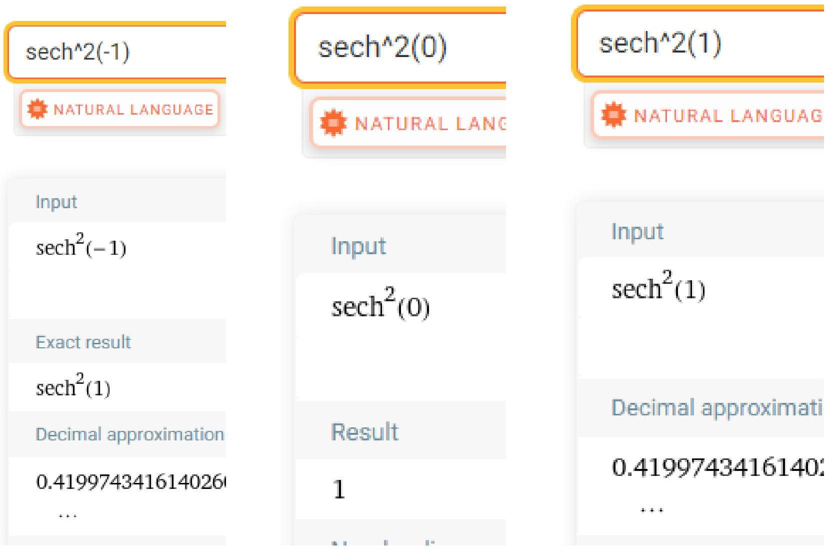
```python
print("TanH Derivative:")
print(TanH.Derivative(outputVector))

ReLu = ReLu()
print("ReLu Derivative:")
print(ReLu.Derivative(outputVector))

LeakyReLu = LeakyReLu()
print("LeakyReLu Derivative:")
print(LeakyReLu.Derivative(outputVector))
```

The output of the above code:

```
Sigmoid Derivative:
['0.19661193324148185']
['0.25']
['0.19661193324148185']

TanH Derivative:
['0.41997434161402614']
['1.0']
['0.41997434161402614']

ReLu Derivative:
['0']
['1']
['1']

LeakyReLu Derivative:
['0.1']
['1']
['1']
```

The WolframAlpha Sigmoid Derivative Results:

The WolframAlpha TanH Derivative Results:



The ReLu Derivative Results (Not supported by WolframAlpha):

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

The Leaky ReLu Results (Not supported by WolframAlpha):

$$\begin{aligned}
\text{LeakyReLu}'(x) &= \begin{cases} x < 0 & 0.1 \\ x >= 0 & 1 \end{cases} \\
\text{LeakyReLu}'(-1) &= 0.1 \\
\text{LeakyReLu}'(0) &= 0 \\
\text{LeakyReLu}'(1) &= 1
\end{aligned}$$

3.2.4 Data Logger Evidence

Evidence 4.1

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

```
inputList = [[random.randint(-10,10), random.randint(-10,10)] for i in range(5)]
print("Unsorted List:")
for item in inputList:
print(item)

dl = DataCollector("SortingTest", [int, int], False)

dl.LogDataPointBatch(inputList)

sortedList = dl.HeapSort(0)

print("Sorted List:")
for item in sortedList:
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

```python
dl = DataCollector("AddPointTest", [int, int], False)
print("Before: ", dl.dataPoints)

dl.LogDataPoint([5, 2])

print("After: ", dl.dataPoints)
```

The output of the above code:

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

Evidence 4.3

Test Data Point matches struture

```python
dl = DataCollector("Match Single Types", [int, float], False)

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

```python
dl = DataCollector("Match Multi Typed", [bool, [float, int]], False)

print("Matches Structure: ", dl.CheckMatchStructure([False, 4.5]))
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

```
dl = DataCollector("Match List Type", [bool, str], False)

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

```
try:
dl = DataCollector("Match Data Structure Error", [str, int], False)

print("Matches Structure: ", dl.CheckMatchStructure(["Steve Preston", True]))
except Exception as x:
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

```
inputList = [[random.randint(-10,10), random.randint(-10,10)] for i in range(5)]
print("Random List:")
for item in inputList:
print(item)

dl = DataCollector("Select List", [int, int], False)

dl.LogDataPointBatch(inputList)

sortedList = dl.Select(0, [1,2,3,5,7])

print("Selected List:")
for item in sortedList:
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

```
inputList = [[random.randint(-10,10), random.randint(-10,10)] for i in range(5)]
print("Saved List:")
```

```python
for item in inputList:
print(item)

dl = DataCollector("Save-Load Test", [int, int], False)

dl.LogDataPointBatch(inputList)

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"

```
DataLogger
  Save-Load Test.data
```

Evidence 4.9

Test for loading a file

```python
dl = DataCollector("Save-Load Test", [int, int], True)

print("Loaded List:")
for item in dl.dataPoints:
print(item)
```

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

```
DataLogger
  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 0x0000
026F66AB19D0>, <enemy.Enemy object at 0x0000026F66AB1A30>, <enemy.Enemy ob
ject 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 0x0000020C7F72B8B0>']
['<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']
```

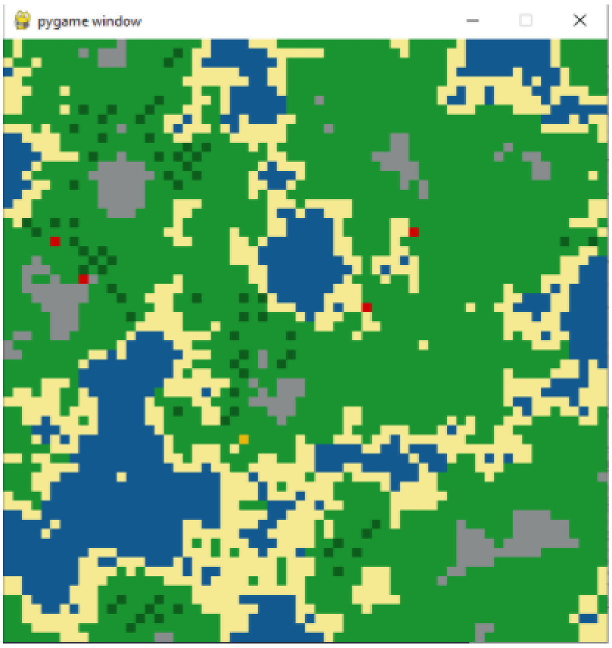
Evidence 5.6

Reward on Left, and the chosen action on Right

```
-0.1 5
```

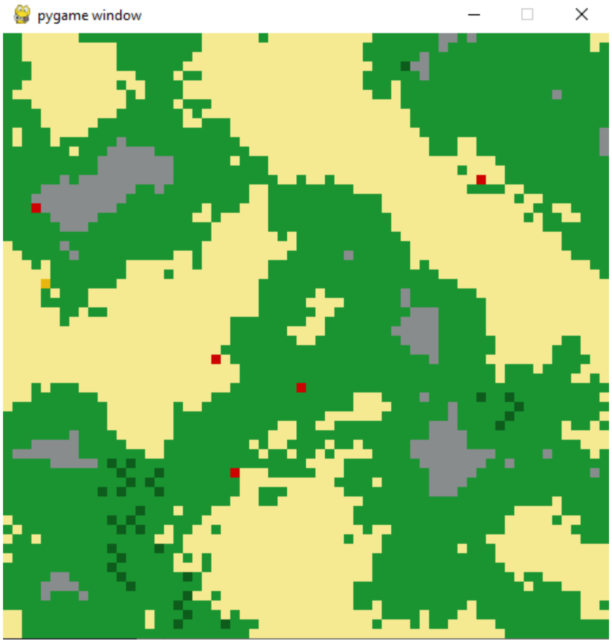
Evidence 5.7

World Generated, is of an Acceptable Standard



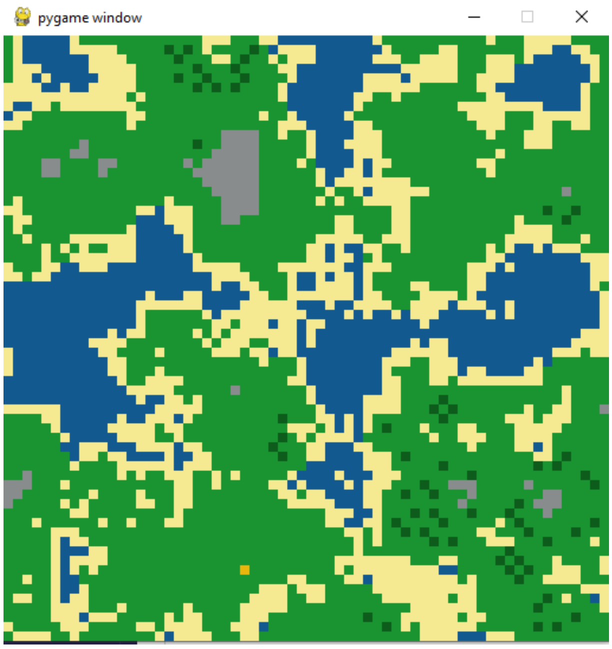
Evidence 5.8

Changed Water Value creates no Water and more Beach

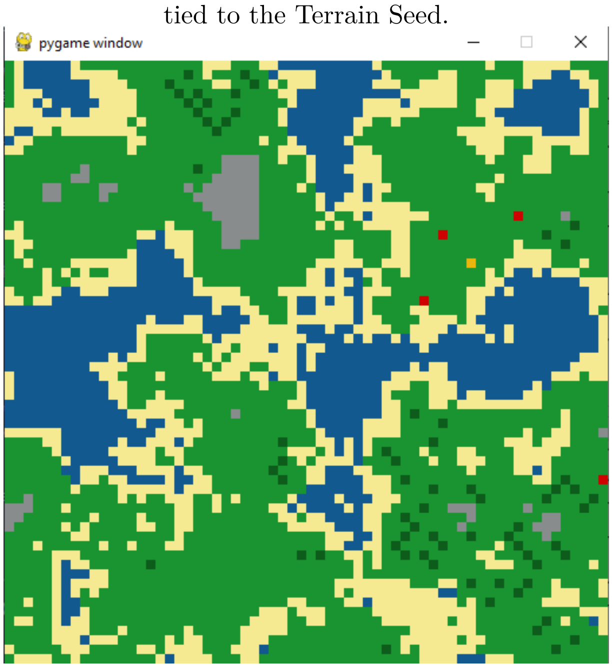


Evidence 5.9

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

Below I have outlined how I have completed each objective that I created as part of my Analyis Section.

1.1 The User can adjust Parameters before the program is run

1.2 There are plenty of Parameters the User can individualy adjust to obtain different results

1.3 The Parameters are stored in a Plaintext Json Formatted File

1.4 The Parameters are checked against individual ranges for each Parameter

1.5 The Ranges are stored in a file seperate to the Parameters

1.6 An Exception is thrown if a Parameter is not within the stored ranges

1.7 The User is prompted to load previous training data from a stored .dqn file

2.1 A Graphical Window is opened after the user input is complete

2.2 This Window displays the current state of the simulation, all the Terrain colours are displayed correctly, mapping to their appropriate height values

2.3 The Agent is Displayed to the Window, and is updated every Action

2.4 The Enemies are displayed to the Window, and is updated every Action

2.5 The Objects are displayed to the Window

2.6 All colours are displayed correctly

2.7 When enabled, the debug side bar is displayed to the right of the Window

2.8 The debug side bar shows the activation values of the Neural Network

3.1 The Data for a created Matrix is stored in a 2d List

3.2 You can retrieve the `order` property from a Matrix, stored as a Tuple<int,int>

3.3.1 You can create a Matrix using a Tuple of Integers

3.3.2 You can create a Matrix using an existing 2d List

3.3.1 You can create a 1 Wide Matrix/Vector using an existing List

3.4 You can fill a Matrix with Randomized Values by specifying `random=True` in the key arguments/kargs

3.5 The Identity Matrix can be created by specifying `identify=True` in the key arguments/kargs

3.6 A Matrix can be printed to the console using a string overload, it is nicely formatted on multiple lines

3.7 Arithmetic Matrix Operations have been implemented

3.7.1 There is a Matrix Addition Operation

3.7.2 There is a Matrix Subtraction Operation

3.7.3 There is a Matrix Multiplication Operation

3.7.4 There is a Matrix Scalar Multiplication Operation

3.7.5 There is a Matrix Hadamard Product Operation

3.7.6 There is a Matrix Transpose Operation

3.7.7 There is a Matrix Sum Operation

3.8 Operator Overloading is used for most of these operations (excluding Tranpose and Sum)

3.9 Efficient Algorithms are utilised where possible for Maximum speed

3.10 A number of Exceptions are in place for when Matrices are used incorrectly

4.1 The generated Terrain is stored within an instance WorldMap, within the tileArray property

4.2 The Terrain and Simulation Data is procedurally generated

4.2.1 The Terrain Generation is based upon an initial seed

4.2.2 The Height Map for the Terrain utilises the Perlin Noise Algorithm

4.2.3 Each Tile of the Terrain contains the individual Height, Type, Colour and Object Data

4.2.4 Tile Types are assigned based on the values specified in the Parameters File

4.2.5 The Colour of each Tile Type is specified by the user in the Parameters File

4.2.6 There are Objects placed within the Simulation

4.2.7 The Objects are placed by the Poisson Disc Sampling Algorithm

4.2.8 Enemies are spawned at Random positions within the Simulation at the start of a new World

4.2.9 The Enemies use a basic pathfinding Algorithm to move towards the Agent each Step

4.3 The Simulation has Rigid States with Specific Actions between them

4.4 A list of possible spawn positions is Generated at the Start of a new World and a random sample is taken from this list which is chosen as the Agents Spawn Position

4.5 The Agents Actions are directly controlled by the Deep Q Learning Algorithm's Outputs

4.6 The Agent has 7 Possible Actions [Up, Right, Down, Left, Interact, Attack, Noop]

4.7 The Agents position can be altered by Performing one of [Up, Right, Down, Left]

4.8 The Agent is able to collect the Items within the simulation through use of the Action [Interact]

4.9 When an Item is collected it is stored in the Agents inventory property

4.10 The Agent can Attack and kill Enemies through the used of the Action [Attack]

4.11 The Agent will be killed when an Enemy is within the same Tile as it

4.12 The Agent will be killed when it is located on a Water Tile

4.13 The Simulation will Reset when the Agent is killed

4.14 The Agent has a Reward Structure which is designed to motivate Exploration and the Gathering of Items

4.14.1 The Agent gains Reward from doing specific Tasks

4.14.2 When the Agent enters a previously unvisited Tile it gains the Exploration Reward specified by the User

4.14.3 When the Agent manages to kill an Enemy through the use of the [Attack] Action, it gains the Attack Reward specified by the User

4.14.4 When the Agent collects one of the generated Items through the use of the [Interact] Action, it gains the Collect Item Reward specified by the User

4.14.5 When the Agent is killed, by walking into Water or by an Enemy, it loses the Death Penalty specified by the User

4.14.6 When the Agent performs the [Attack] Action but doesnt manage to Kill an Enemy with it, it loses the Failed Attack Penalty specified by the User

4.15 When the Agent is killed and the Simulation Resets, the Terrain is regenerated using the same Procedural Method (Perlin Noise)

4.16 When the Agent is killed and the Simulation Resets, the Objects are regenerated using the same Procedural Method (Poisson Disc Sampling)

4.17 When the Agent is killed and the Simulation Resets, the Enemies are spawned again using the same method

4.18 When the Agent is killed and the Simulation Resets, the Agents Spawn Position is regenerated and set

4.19 The Agent has the Method GetTileVector to sample it's surrounding Tile Data

4.20 The Agent has the Method TileVectorPostProcess to convert the Tile Objects into Grayscale Values

4.21 The Grayscale Values are utilised by the Deep Q Learning Algorithm as it's Input for Forward Propagation

5.1 The Dual Neural Network Class has two properties mainNetwork and targetNetwork which are both instances of the Neural Network Class

5.1.1 The Neural Network Class utilises an Object Orientated Model by holding a list of Layer Objects which store Matrices pertaining to the Weights and Biases of the Network

5.1.2 The Neural Network Class contains the Method ForwardPropagation which takes an Input Vector of Grayscale Values and Propagates them through the Network leaving Unactivated and Activated Values in each Layer Object

5.1.3 The Neural Network Class calculates the Expected Value for Half Square Difference utilising the Bellman Equation to determine the payoff of each action and its future Reward

5.1.4 The Expected Values are used to calculate the Loss of the Networks Output from Forward Propagation

5.1.5 The Neural Network Class contains the Method BackPropagation which takes a Vector of Loss Values per Network Output, and Back Propagates this through the Network by calculating derivatives per weight and bias

5.1.6 The Calculated Loss is used for the BackPropagation Method

5.1.7 Both ForwardPropagation and BackPropagation utilise purely Matrix Operations in order to achieve their results

5.1.8 Activation Functions are specified as part of the Neural Network and the Normal and Derivatives are used as part of the ForwardPropagation and BackPropagation Methods

5.2 The Target Networks Weights are updated intermittently, the number of steps between updates is specified by the User in the Parameters File

5.3 Grayscale Values are inputted into the Network after being collected and processed by the Agent

5.4 The Network calls the Agents Methods GetTileVector and TileVectorPostProcess to get the Grayscale Data

5.5 The Neural Networks Output is used to inform the Agents next Action

5.6 When selecting an Action the Network utilises the Epsilon Greedy decision process, it generates a Random Number from $0 \rightarrow 1$, if this values is less than the current Epsilon Value it will pick a Random Action instead of the Action Informed from the Network

5.7 If a Random Action is not selected, the Agent will pick an Action based upon the distribution generated by the SoftMax Function, actions with a higher probability are more likely to be picked

5.8 Experience Replay is used to learn from past Experiences

5.9 Each State Action Pair is stored to be learned from in the future

5.10 Experience Replay is performed every N steps, specified by the User in the Parameters File

5.11 The Weights and Biases of the Network, along with other Elements are stored to .dqn Files, so that if the User wishes to Resume the Training of the Network at another time they can do that

5.12 The Experience Replay State Action Pairs are stored in a Double Ended Queue Object

6.1 The Defined Activation Functions are Utilised within the Neural Networks ForwardPropagation and BackPropagation Methods

6.2 An Activation is defined as an Abstract Base Class

6.3 The Abstract Base Class contrains Activation and Derivative Methods which require Implementations

6.4 Both Methods are designed to take Vectors as Inputs

6.5 Both Methods are deisgned to return Vectors as Outputs

6.6.1 Sigmoid is Implemented as an Activation Function

6.6.2 TanH is Implemented as an Activation Function

6.6.3 ReLu is Implemented as an Activation Function

6.6.4 Leaky ReLu is Implemented as an Activation Function

7.1 When creating a Data Logger within the code, a Data Structure must be specified as a list of Types

7.2 When a Data Point is added it is checked against the Loggers Structure to make sure it can be plotted correctly

7.3 If it does not match the Structure an Exception will be thrown stating which part of the structure it does not match

7.4 The Data Points can be saved to a .data File with the SaveDataPoints Method

7.5 There is a Heap Sort Implementation Utilising a Heap Data Structure, which can be used to sort in Ascending or Descending Order

7.6 When Heap Sorting you can sort Data by an index in the Data Points

7.7 The Heap Sort is implemented utilising the nlog(n) algorithm

8 There is a Script which allows the User to plot .data files to a MatPlotLib Graph

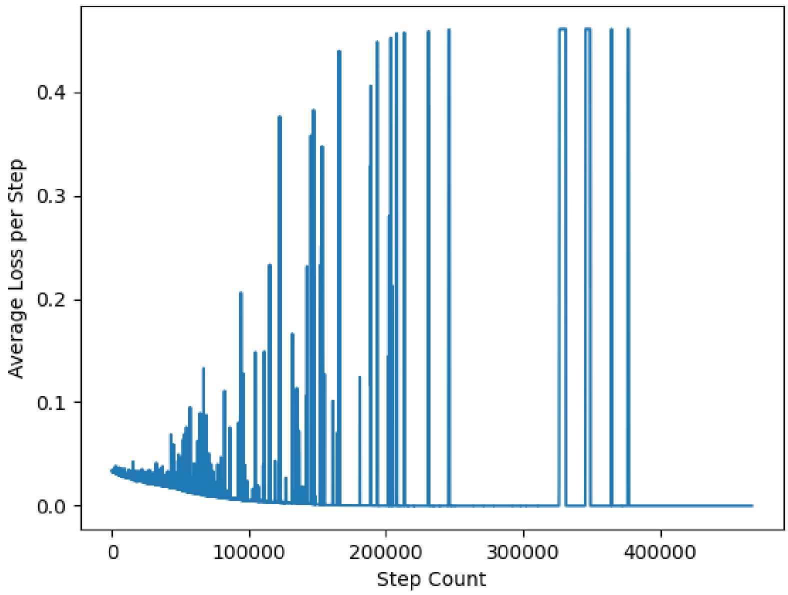
8.1 Upon running the Script the User is presented with a List of .data Files in the DataLogger Directory

8.2 The User is asked to specify which Points of the Data they want to plot

8.3 The Graph is displayed after the User Input is Complete

4.2 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.

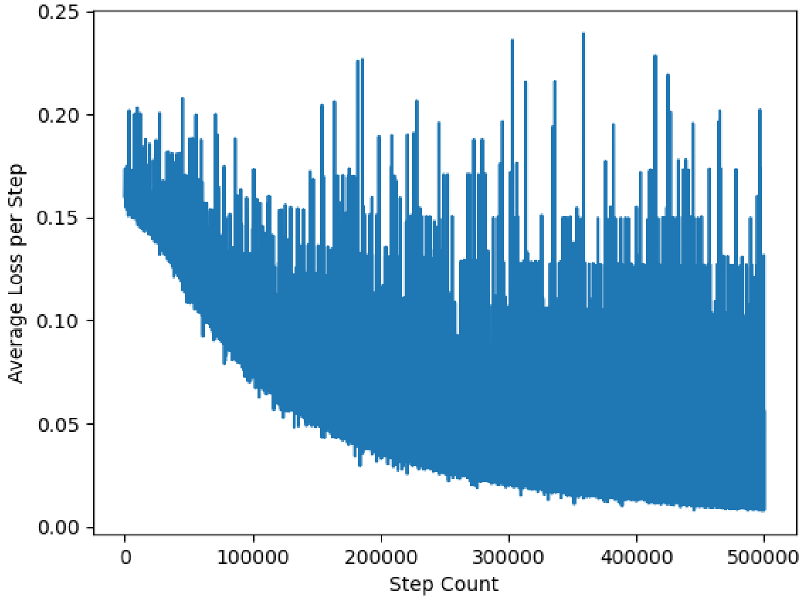


Neural Network flatlining towards 0 loss by only picking ”Noop”

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 doesnt manage to overall minimise it. I believe this is a sign that the simulation is too complex for the Network architecture to solve.

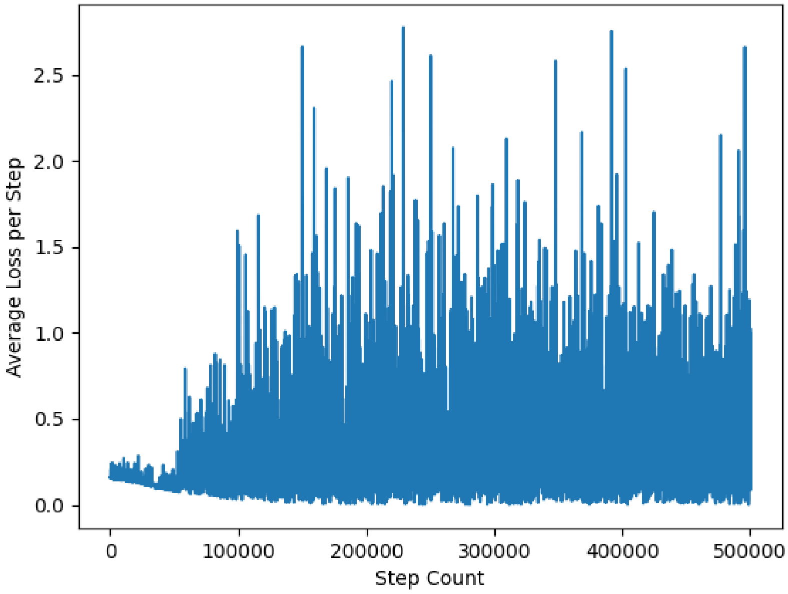


Neural Network attempts to minimise network but fails to solve the simulation

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 signifcance 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.

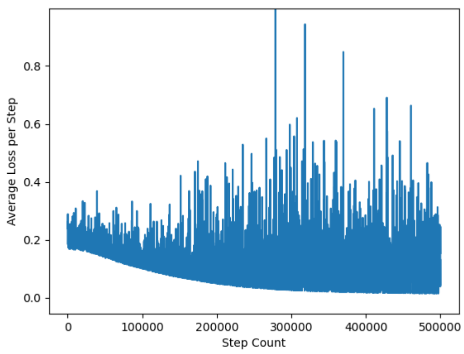


Neural Network struggles with Enemies

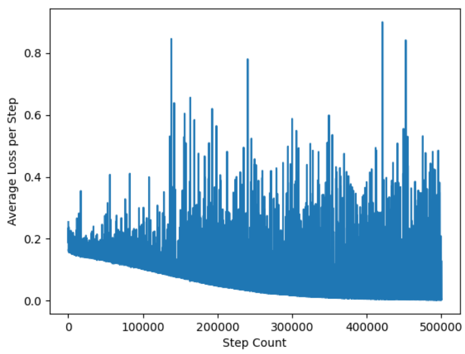
Large network architecture with 49 Input Nodes

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.

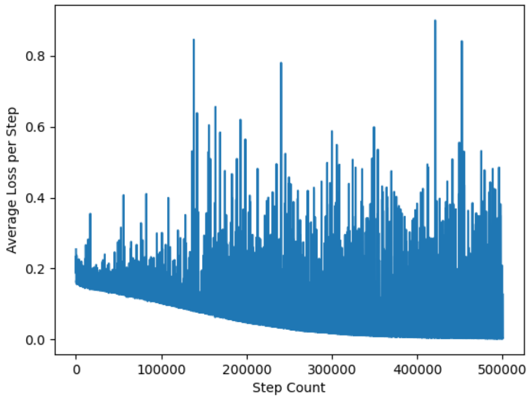


(p) 25 Input Nodes

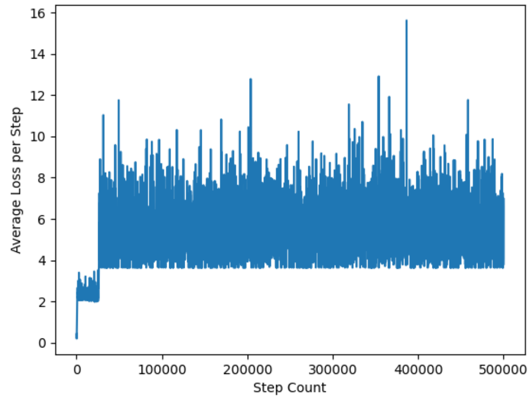


(q) 9 Input Nodes

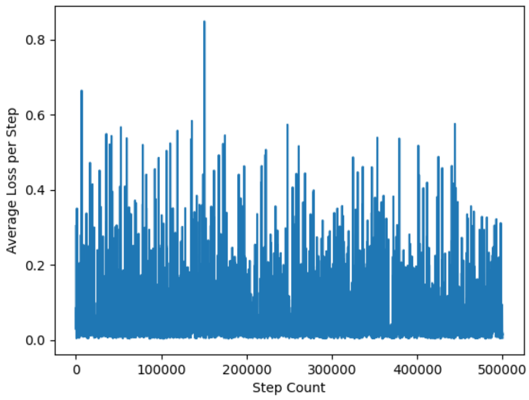
I then chose the best performing Network out of the 3 I tested, which has 25 Inputs, and tried it with all the Activation Functions I've implemented. Previously I had just been using the standard Sigmoid Activation funtion. This is an attempt to find the best possible Network ⇌ Activation Combination.



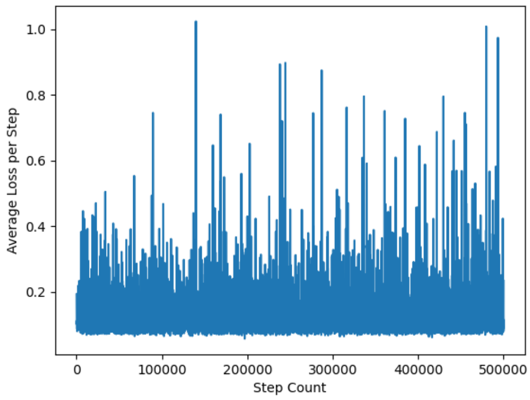
(r) Sigmoid Activation Function



(s) TanH Activation Function



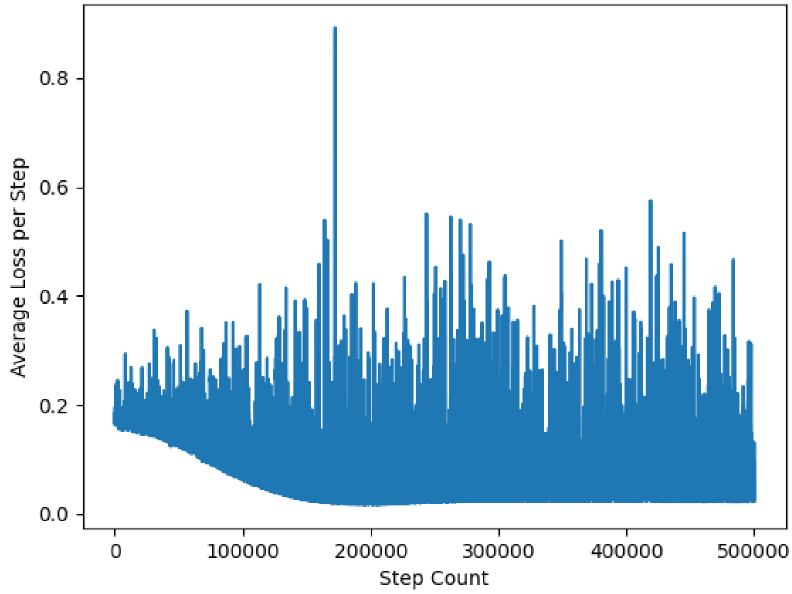
(t) ReLu Activation Function



(u) 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 preform 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.

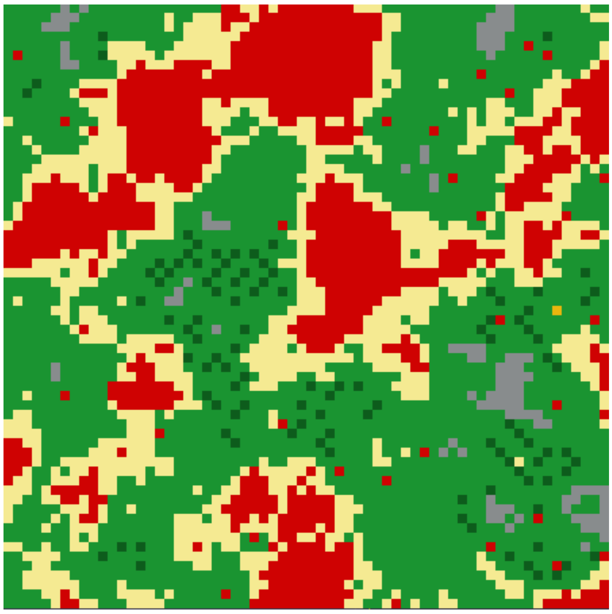
In an attempt to reduce the complexity of the simulation I created, I altered the simulation slightly. I turned off the Enemies movement, this was an attempt to reduce the difficulty of the problem for the Agent. I also spawned 30 rather than 5 enemies at the start of a created world. This resulted in somewhat better results when compared to previous results, the baseline loss minimises towards 0 quicker. This is definitely because the Network finds static threats easier to deal with. I also noticed in the previous test that the Agent didn't appear to have a problem avoiding the Static Enemies, so I kept this for the next test.



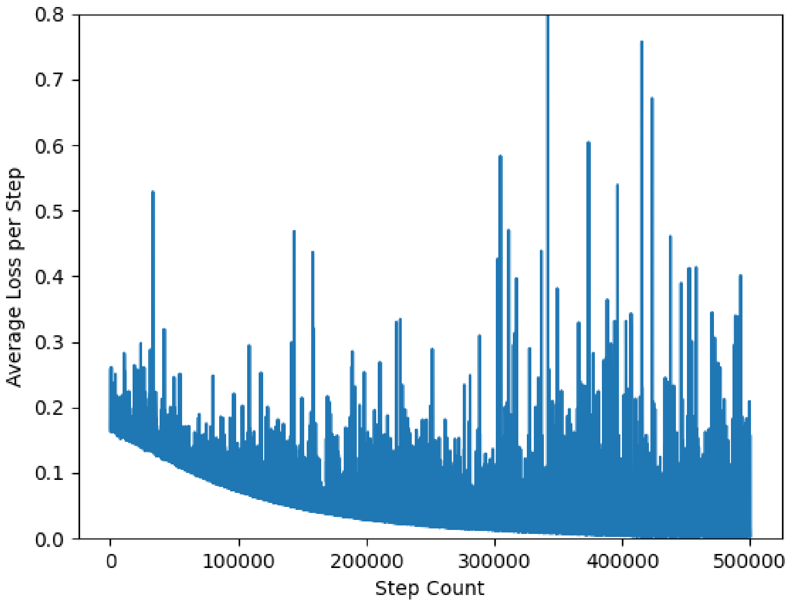
Altered Simulation Data using Static Enemies

25 Input Nodes

This next test involved me changing the Colour of the Water to the same colour as the Enemies. I figured that this would improve the Networks Ability to determine what is a threat to its Survival. Instead of having to form a relationship between two colours, it was only one. This performed quite well in comparison to previous tests. The overall average loss is less than every other test, and shows that the Network is actually capable of determining relationships between the input values and correct outputs.



Altered Simulation using Static Enemies and Red Water



Simulation Data

25 Input Nodes

4.3 Answering the Proposed Questions

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 fully solve the simulation. After being trained to 500000 steps with multiple different layer structures and parameters, it fails to achieve a true solution to the problem at any given time step. The Average Loss of of the Network for most of my tests trends downwards, but still remains highly innacurate. The graphed data shows that the Average Loss (Plotted per 100 Steps) constantly peaks and drops back down to the baseline. This baseline for most of the tests I performed trends downwards as a curve. All the Tests Data is shown above.

Because the implemented algorithm has not managed to fully solve the problem, I have answered the sub-questions I outlined in my *Statement of Investigation*:

- The Algorithm quite clearly forms a link between specific elements in the simulation and danger, even if it doesnt manage to avoid them always. This can be shown by the Network managing to identify and kill the Enemies, along with performing better in my test where I altered the Colour of the Water to be the same as the enemies colour. With this in place the Network manages to perform better, showing a clear link between the inputted colour Red and the danger associated with it. This answers the 1st sub-question, *Does the Algorithm draw links between specific elements and danger?*.

- The Algorithm does manage to pickup the occasional item when attempting to solve the simulation. But it is unclear wether this is by random chance or if this is the intended action of the Algorithm. There is not enough evidence to suggest that the Algorithm can perform well with specific collection tasks of the items in the Simulation. Therefore answering the 2nd sub-question, *How well does the Algorithm perform with specific tasks?*.

- I tested the Network with different Activation Functions and Layer Structures, I found that some tests performed better than others. This shows that the Algorithm can perform better when tuning the parameters, answering the 4th sub-question I proposed, *"Can I fine tune the Algorithms Parameters to get better results?"*.

- I performed tests where I altered the simulation in order to reduce the complexity for the Algorithm. This included making the Enemies Static, and changing the Colour of the Water. Both of these appeared to show the Average Loss of the Network Decrease at a faster rate, and a reduction of peaks in the Average Loss. The Water Colour change appeared to show the best Training results out of all the test results. This shows that when reducing the complexity of the problem, the Algorithm manages to better solve the given problem, answering the 3rd sub-question I proposed, *"If the problem is altered to be simpler does the Algorithm perform better?"*.

Overall the Algorithm implemented shows it can solve individual parts of the problem, but when combined together the complexity is too much for it to solve completely. I believe that the main problem here is the generalisation needed to solve a pseudorandomly generated environment. If the Algorithm was facing the exact same problem each time, with a linear path forward, it would most likely have more success when attempting to solve that problem.

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 neccesary.

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 alot 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 ontop 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, youve 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 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 (Turns out it was because of the complexity of the problem). This help was incredibly valuable in completing my Technically 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 Analyis. 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 kindof dissapointing as well that the Network has been failed to truly 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 interpretted language it is comparatively slow compared to the other programming languages I considered using. C is a compiled language so it is comparatively alot 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 wouldnt be able to solve these problems effectively though without the implementation of my first improvement, a Convolutional Neural Network.

5 Technical Solution

This section includes all the code relating to my Technical Solution, along with the two Json Formatted files which are used within my project.

5.1 main.py

```
import pygame
from simulation import *
import time

params = Simulation.LoadParameters("Default") # Loads parameters
Simulation.CheckParameters(params, "Range") # Checks parameters

gameSim = Simulation(params) # Create and initiate simulation
gameSim.InitiateSimulation()

# Creates pygame window - includes side debug offset if needed
worldResolution = params["WorldSize"] * params["TileWidth"]
if params["Debug"]:
    debugOffset = (len(params["DeepQLearningLayers"]) * params["TileWidth"] * params["DebugScale"])
else:
    debugOffset = 0
window = pygame.display.set_mode((worldResolution + debugOffset, worldResolution))
pygame.display.set_caption('Comp Sci NEA')

stepDelay = params["StepDelay"] # Time step Delay

# Constant loop running
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT: # If window exit than close end program
            running = False
```

```
            if event.type == pygame.KEYDOWN: # Key Down
                if event.key == pygame.K_F1: # Force Create new world
                    gameSim.CreateWorld()
                if event.key == pygame.K_F2: # Force Kill agent
                    gameSim.agent.alive = False

        if gameSim.step > params["DQLearningMaxSteps"]:
            running = False

        gameSim.TimeStep() # Perform a timestep
        time.sleep(stepDelay) # Sleep if needed

        gameSim.RenderToCanvas(window) # Draw to canvas

        pygame.display.update() # Update pygame window to display content
```

5.2 simulation.py

```
from worldClass import *
from newAgent import *
from enemy import *
from deepqlearning import *
import random, pygame, math

# Interface class between Main and Every other class
class Simulation():
    def __init__(self, params): # Constructor for Simulation
        self.paramDictionary = params

        self.worldMap = None
        self.network = None
        self.agent = None

        self.enemyList = []

        self.step = 0

# Step forward network methods
    def TimeStep(self): # Steps forward 1 cycle
        if not self.agent.alive: # Resets Sim if Agent is dead
            self.ResetOnDeath()

        self.network.TakeStep(self.agent, self.worldMap, self.enemyList) # Take step with Deep Q
            ↪ Network

        if self.paramDictionary["EnableEnemies"]: # If enemies enabled then update enemies
            self.UpdateEnemies()

        self.step += 1

    def UpdateEnemies(self): # Updates Enemies
        self.enemyList = [x for x in self.enemyList if x is not None] # Clears None type from list

```

```
        for i in range(len(self.enemyList)): # Commits each Enemies actions and sets to None if they
        ↪ died in that step
            self.enemyList[i].CommitAction(self.agent, self.worldMap)

            if not self.enemyList[i].alive: # Removes dead enemies from list
                self.enemyList[i] = None

        self.enemyList = [x for x in self.enemyList if x is not None] # Clears None type from list

# Creation and Initialisation Methods
    def InitiateSimulation(self): # Initialises Simulation
        self.CreateWorld()
        self.CreateAgent()

        self.CreateDeepQNetwork()

    def CreateWorld(self, seed = 0): # Creates new world with specified or random seed
        if seed == 0: seed = random.randint(0, 999999)

        if self.worldMap == None: # Creates a new world map if one does not exist - otherwise resets
        ↪ the seed
            self.worldMap = WorldMap(seed, self.paramDictionary)
        else:
            self.worldMap.MAP_SEED = seed

        if self.paramDictionary["GenerateThreaded"]: # Generates Terrain using 4 threads if
        ↪ specified
            self.worldMap.GenerateThreadedParent()
        else:
            self.worldMap.GenerateMap()

        self.worldMap.GenerateTreeArea() # Generates Tree Area

        self.worldMap.RenderMap() # Renders Map and Renders Interactables
        self.worldMap.RenderInteractables()

        if self.paramDictionary["EnableEnemies"]: # Spawns Enemies if specified
            self.SpawnEnemies()

        print("Created New World, Seed: {}".format(seed))

    def CreateDeepQNetwork(self, layers = None): # Creates a Deep Q Network with the given Hyper
    ↪ Parameters
        if layers == None:
            layers = self.paramDictionary["DeepQLearningLayers"]

        if self.network == None: # Creates a Network if one doesnt already exist
            if self.paramDictionary["EnterValues"]:
                load = input("Load weights (Y/N): ")
                if load.upper() == "Y":
                    fName = input("State file name: ")

                    self.network = DoubleNeuralNet(layers, self.paramDictionary, load=True,
                    ↪ loadName=fName)
                else:
```

```python
                    self.network = DoubleNeuralNet(layers, self.paramDictionary)
            else:
                self.network = DoubleNeuralNet(layers, self.paramDictionary)

    def CreateAgent(self): # Creates an agent / Resets existing agent
        if self.agent == None:
            self.agent = Agent(Agent.SpawnPosition(self.worldMap), self.paramDictionary)
        else:
            self.agent.Reset(self.worldMap)

    def SpawnEnemies(self, n = 0): # Spawns <= n enemies on call
        if n == 0: n = self.paramDictionary["StartEnemyCount"]

        for count in range(n): # Spawns enemies for count
            spawnLoc = Enemy.SpawnPosition(self.worldMap, self.enemyList)
            if spawnLoc == None:
                continue
            else:
                tempEnemy = Enemy(spawnLoc, self.paramDictionary)
                self.enemyList.append(tempEnemy)

    def ResetOnDeath(self): # Resets Simulation if Agent Dies
        self.CreateWorld()
        self.CreateAgent()
        self.enemyList = []

        if self.paramDictionary["EnableEnemies"]: # Spawns Enemies if specified
            self.SpawnEnemies()

# Render Methods
    def RenderToCanvas(self, window): # Render Content to Canvas
        TW = self.paramDictionary["TileWidth"]
        DS = self.paramDictionary["DebugScale"]

        if self.paramDictionary["Debug"]: # Renders debug info for Neural Network if specified
            for i in range(len(self.network.MainNetwork.layers)):
                for k in range(self.network.MainNetwork.layers[i].activations.order[0]):
                    value = self.network.MainNetwork.layers[i].activations.matrixVals[k][0]
                    newVal = (math.tanh(value) + 1) / 2
                    colourTuple = (255 * newVal, 255 * newVal, 255 * newVal)

                    try: # Exceps if colour value out of range
                        pygame.draw.rect(window, colourTuple, ((self.paramDictionary["WorldSize"] * TW + i * TW * DS), (k * TW * DS), (TW * DS), (TW * DS)))
                    except:
                        print(newVal)

        self.worldMap.DrawMap(window) # Draws Content to window

        for i in range(len(self.enemyList)): # Draws enemies to window
            pygame.draw.rect(window, self.paramDictionary["ColourEnemy"], ((self.enemyList[i].location[0] * TW), (self.enemyList[i].location[1] * TW), TW, TW))

        # Draws Player to window
```

```
        pygame.draw.rect(window, self.paramDictionary["ColourPlayer"], ((self.agent.location[0] * TW), (self.agent.location[1] * TW), TW, TW))

# Miscellaneous Methods
    @staticmethod
    def LoadParameters(fname): # Load Parameters from file and store them in a dictionary
        file = open("Parameters\\{}.param".format(fname), "r")
        params = json.loads(file.read())
        file.close()
        return params

    @staticmethod
    def CheckParameters(params, fname): # Checks every parameter against the range.parm file
        file = open("Parameters\\{}.param".format(fname), "r") # Read range file
        paramRanges = json.loads(file.read()) # Load with json module
        file.close()

        for param in params: # Checks if parameter is specified in range file - If specified than check against given value to check within range
            if param in paramRanges:
                valRange = paramRanges[param]
                val = params[param]

                if valRange[1] == None: pass
                elif val > valRange[1]:
                    raise Exception("'{}' of value {}, has exceeded the range: {}-{}".format(param, val, valRange[0], valRange[1])) # Greater than specified range

                if valRange[1] == None: pass
                elif val < valRange[0]:
                    raise Exception("'{}' of value {}, has subceeded the range: {}-{}".format(param, val, valRange[0], valRange[1])) # Less than specified range

        print("Parameters within Specified Ranges")
```

5.3 newAgent.py

```
from worldClass import *
from random import shuffle
from matrix import Matrix
from copy import copy

class Agent():
    def __init__(self, location, params):
        self.paramDictionary = params

        self.location = location

        self.alive = True

        self.emptyInventory = {"Wood": 0}
        self.inventory = self.emptyInventory

# Methods for tile vectors
    def GetTileVector(self, worldMap, enemyList): # Returns a Vector of Tile Datatype
```

```
        offset = self.paramDictionary["DQLOffset"]
        sideLength = 2 * offset + 1
        tileVec = Matrix((sideLength * sideLength, 1))

        blankOceanTile = Tile()
        blankOceanTile.InitValues(0, 0, self.paramDictionary["ColourWater"]) # Blank ocean tile for edge case

        enemyLocList = [enemyList[i].location for i in range(len(enemyList)) if enemyList[i] is not None]

        n = 0
        for y in range(self.location[1] - offset, self.location[1] + offset + 1): # Loop through Tiles in surrounding area
            for x in range(self.location[0] - offset, self.location[0] + offset + 1):
                if 0 <= x and x <= self.paramDictionary["WorldSize"] - 1 and 0 <= y and y <= self.paramDictionary["WorldSize"] - 1:
                    tileVec.matrixVals[n][0] = copy(worldMap.tileArray[x][y])
                    if [x,y] in enemyLocList:
                        tileVec.matrixVals[n][0].WriteEnemy() # Writes enemies to tile if they exist
                else:
                    tileVec.matrixVals[n][0] = blankOceanTile # Write water tile when out of range of the world - Literal edge case
                n += 1
        return tileVec

    def TileVectorPostProcess(self, tileVec): # Returns 2 Vectors, 1 of tile types, 1 of grayscale values
        tileTypeVec = Matrix(tileVec.order)
        tileGrayscaleVec = Matrix(tileVec.order)

        for n in range(tileVec.order[0]): # Converts vector to grayscale and type vectors
            tileTypeVec.matrixVals[n][0] = tileVec.matrixVals[n][0].tileType

            if tileVec.matrixVals[n][0].hasEnemy: # Enemy will overwrite tile colour if they are within that tile
                tileGrayscaleVec.matrixVals[n][0] = self.ColourToGrayscale(self.paramDictionary["ColourEnemy"])
            else:
                tileGrayscaleVec.matrixVals[n][0] = self.ColourToGrayscale(tileVec.matrixVals[n][0].tileColour)

        return tileTypeVec, tileGrayscaleVec

    def ColourToGrayscale(self, colourTuple): # Converts colour value (255,255,255) to grayscale (0-1)
        grayscale = (0.299 * colourTuple[0] + 0.587 * colourTuple[1] + 0.114 * colourTuple[2]) / 255
        return grayscale

# Action Methods
    def CommitAction(self, action, tileObjVec, worldMap, enemyList): # Commits the given Action
        offset = self.paramDictionary["DQLOffset"]
        sideLength = 2 * offset + 1

        if action == 0:
```

```
            self.Move(action, worldMap) # Move Up

        elif action == 1:
            self.Move(action, worldMap) # Move Right

        elif action == 2:
            self.Move(action, worldMap) # Move Down

        elif action == 3:
            self.Move(action, worldMap) # Move Left

        elif action == 4 and tileObjVec.matrixVals[(sideLength * offset) + offset][0].hasObject == True: # Pickup item
            self.PickupItem(worldMap)

        elif action == 5: # Attack Surroundings
            self.Attack(enemyList)

        elif action == 6: # Noop/Null action
            pass
            #print("Noop")

    def Move(self, direction, worldMap): # Moves agent in given Direction
        if direction == 0: self.location = [self.location[0], self.location[1] - 1] # Move Up
        elif direction == 1: self.location = [self.location[0] + 1, self.location[1]] # Move Right
        elif direction == 2: self.location = [self.location[0], self.location[1] + 1] # Move Down
        elif direction == 3: self.location = [self.location[0] - 1, self.location[1]] # Move Left

        self.alive = self.CheckIfValidStandTile(self.location, worldMap)
        if not self.alive: return

        if worldMap.tileArray[self.location[0]][self.location[1]].explored == False: # Checks if tile is explored or not
            worldMap.tileArray[self.location[0]][self.location[1]].explored = True

    def CheckIfValidStandTile(self, location, worldMap): # Checks if tile will murder the agent
        x = location[0]
        y = location[1]
        if 0 <= x and x <= self.paramDictionary["WorldSize"] - 1 and 0 <= y and y <= self.paramDictionary["WorldSize"] - 1: pass
        else:
            return False

        if worldMap.tileArray[x][y].tileType == 0: # Checks if tile is water
            return False

        return True

    def PickupItem(self, worldMap): # Pickup Item in the same tile as Agent
        if worldMap.tileArray[self.location[0]][self.location[1]].hasObject:
            self.inventory[worldMap.tileArray[self.location[0]][self.location[1]].objectType] += 1

            worldMap.tileArray[self.location[0]][self.location[1]].ClearObject()

    def Attack(self, enemyList): # Attacks in a given Area surrounding Agent
```

```
        enemyLocList = [enemyList[i].location for i in range(len(enemyList))]

        for y in range(self.location[1] - 1, self.location[1] + 2): # Loop through Tiles in
        ↪ surrounding area
            for x in range(self.location[0] - 1, self.location[0] + 2):
                if [x,y] in enemyLocList:
                    for i in range(len(enemyLocList)):
                        if enemyLocList[i] == [x,y]:
                            enemyList[i] = None

        enemyList = [x for x in enemyList if x is not None] # Clears enemy list of None type

# Reward Method
    def GetReward(self, action, tileObjVec): # Gets reward given action and tile vector
        offset = self.paramDictionary["DQLOffset"]
        sideLength = 2 * offset + 1

        cumReward = 0

        if action == 0: # Move Up
            tile = tileObjVec.matrixVals[(sideLength * (offset - 1)) + offset][0]
            cumReward += self.MoveReward(tile)

        elif action == 1: # Move Right
            tile = tileObjVec.matrixVals[(sideLength * offset) + offset + 1][0]
            cumReward += self.MoveReward(tile)

        elif action == 2: # Move Down
            tile = tileObjVec.matrixVals[(sideLength * (offset + 1)) + offset][0]
            cumReward += self.MoveReward(tile)

        elif action == 3: # Move Left
            tile = tileObjVec.matrixVals[(sideLength * offset) + offset - 1][0]
            cumReward += self.MoveReward(tile)

        elif action == 4: # Pickup Item
            if tileObjVec.matrixVals[(sideLength * offset) + offset][0].hasObject:
                cumReward += self.paramDictionary["CollectItemReward"]
            else:
                cumReward += self.paramDictionary["NoopReward"]

        elif action == 5: # Attack
            cumReward += self.CombatReward(tileObjVec)

        elif action == 6: # No action/Noop/Idle
            cumReward += self.paramDictionary["NoopReward"]

        return cumReward

    def MoveReward(self, tileObj): # Gets Reward given Agent moving into a tile
        reward = 0
        if tileObj.tileType == 0 or tileObj.hasEnemy:    # Adds death reward if enemy or water
            reward += self.paramDictionary["DeathReward"]
        else:                                             # Else adds explore and move reward
            if tileObj.explored == False:
```

```
                reward += self.paramDictionary["ExploreReward"]
            reward += self.paramDictionary["MoveReward"]
        return reward

    def CombatReward(self, tileObjVec):
        killReward = self.paramDictionary["AttackReward"]
        offset = self.paramDictionary["DQLOffset"]
        sideLength = 2 * offset + 1

        reward = 0

        # Checks tiles around agent for enemies, adding reward where neccesary
        if tileObjVec.matrixVals[(sideLength * (offset - 1)) + offset - 1][0].hasEnemy: reward +=
        ↪ killReward
        if tileObjVec.matrixVals[(sideLength * (offset - 1)) + offset][0].hasEnemy:     reward +=
        ↪ killReward
        if tileObjVec.matrixVals[(sideLength * (offset - 1)) + offset + 1][0].hasEnemy: reward +=
        ↪ killReward

        if tileObjVec.matrixVals[(sideLength * offset) + offset - 1][0].hasEnemy:       reward +=
        ↪ killReward
        if tileObjVec.matrixVals[(sideLength * offset) + offset][0].hasEnemy:           reward +=
        ↪ killReward
        if tileObjVec.matrixVals[(sideLength * offset) + offset + 1][0].hasEnemy:       reward +=
        ↪ killReward

        if tileObjVec.matrixVals[(sideLength * (offset + 1)) + offset - 1][0].hasEnemy: reward +=
        ↪ killReward
        if tileObjVec.matrixVals[(sideLength * (offset + 1)) + offset][0].hasEnemy:     reward +=
        ↪ killReward
        if tileObjVec.matrixVals[(sideLength * (offset + 1)) + offset + 1][0].hasEnemy: reward +=
        ↪ killReward

        if reward > 0: return reward
        else: return self.paramDictionary["AttackFailedReward"]

    def GetRewardVector(self, tileObjVec, outputs): # Returns Vector of Reward Values Per action
        returnVec = Matrix((outputs, 1))

        for i in range(outputs):
            returnVec.matrixVals[i][0] = self.GetReward(i, tileObjVec)

        return returnVec

    def MaxQ(self, rewardVec): # Used to get Max Reward from reward Vector
        return max([rewardVec.matrixVals[i][0] for i in range(rewardVec.order[0])]) # Utilises List
        ↪ Comprehension

# Miscellaneous Methods
    def Reset(self, worldMap): # Resets Inventory and Location of Agent
        self.inventory = self.emptyInventory

        self.location = Agent.SpawnPosition(worldMap)

        self.alive = True
```

```

    @staticmethod
    def SpawnPosition(worldMap): # Returns a coord in which the Agent can spawn
        spawnList = []

        for y in range(0, worldMap.MAP_SIZE):
            for x in range(0, worldMap.MAP_SIZE):
                if worldMap.tileArray[x][y].tileType == 2:
                    spawnList.append([x, y])

        shuffle(spawnList)
        return spawnList[0]
```

5.4 enemy.py

```
from newAgent import *
from random import randint

class Enemy(Agent): # Enemy inherits from Agent Class
    def __init__(self, location, params): # Constructor for Enemy Class
        self.paramDictionary = params

        self.location = location

        self.alive = True

    def CommitAction(self, agent, worldMap): # Override of Agent Class method
        xDif = agent.location[0] - self.location[0]
        yDif = agent.location[1] - self.location[1]

        if xDif == 0 and yDif == 0: # Checks if on Agent - If so -> Kill Agent
            agent.alive = False
            return

        # Basic Path Finding for enemy
        # Calculates difference between agent and player position, and moves in the greatest
        ↪ direction
        if abs(xDif) > abs(yDif):  # X Dif > Y Dif
            if xDif > 0:
                self.location[0] += 1
            else:
                self.location[0] -= 1
        elif abs(xDif) < abs(yDif): # Y Dif > X Dif
            if yDif > 0:
                self.location[1] += 1
            else:
                self.location[1] -= 1
        else:                       # Move random direction when X Dif = Y Dif
            if randint(0,1):
                if xDif > 0:
                    self.location[0] += 1
                else:
                    self.location[0] -= 1
            else:
                if yDif > 0:
```

```
                    self.location[1] += 1
                else:
                    self.location[1] -= 1

        self.alive = self.CheckIfValidStandTile(self.location, worldMap) # Checks if walked into water or not

    @staticmethod
    def SpawnPosition(worldMap, enemyList): # Generate spawn position for the enemy given worldMap and enemyList - Static method
        spawnList = []
        enemyLocList = [enemyList[i].location for i in range(len(enemyList))]

        for y in range(0, worldMap.MAP_SIZE):
            for x in range(0, worldMap.MAP_SIZE):
                if worldMap.tileArray[x][y].tileType == 2: # Checks if tile type is
                    spawnList.append([x, y])

        shuffle(spawnList)

        if spawnList[0] in enemyLocList: # Select spawn if not already selected
            return None
        else:
            return spawnList[0]
```

5.5 worldClass.py

```
import json, random, pygame, threading
import perlinNoise

# Class to store Individual Tile Data
class Tile():
    def __init__(self): # Initialise Tile object
        self.tileHeight = -1
        self.tileType = 0
        self.tileColour = (0,0,0)
        self.explored = False
        self.hasObject = False
        self.hasEnemy = False

    def InitValues(self, tileType, height, colour): # Set/Initialise Tile Vales
        self.tileType = tileType
        self.tileHeight = height
        self.tileColour = colour

    def AddObject(self, objectType, objectColour): # Adds an Object to the Tile Object
        self.hasObject = True
        self.objectType = objectType
        self.objectColour = objectColour

    def ClearObject(self): # Clears Object from the Tile Object
        self.hasObject = False
        self.objectType = ""
        self.objectColour = (0,0,0)

```

```python
    def WriteEnemy(self): # Write Enemy to tile
        self.hasEnemy = True

# Class to store world terrain and object data
class WorldMap():
    def __init__(self, seed, params): # Initialise method for creating an instance of the world
        self.MAP_SIZE = params["WorldSize"]
        self.TILE_WIDTH = params["TileWidth"]
        self.MAP_SEED = seed
        self.TILE_BORDER = params["TileBorder"]

        self.tileArray = [[Tile() for i in range(self.MAP_SIZE)] for j in range(self.MAP_SIZE)]

        self.paramDictionary = params

# Non Threaded Terrain Generation
    def GenerateMap(self): # Generates terrain - Not Threaded
        for y in range(0, self.MAP_SIZE):
            for x in range(0, self.MAP_SIZE):
                xCoord = x / self.MAP_SIZE * self.paramDictionary["WorldScale"]
                yCoord = y / self.MAP_SIZE * self.paramDictionary["WorldScale"]

                self.tileArray[x][y].tileHeight = perlinNoise.OctaveNoise(self.MAP_SEED + xCoord, self.MAP_SEED + yCoord,
                                                                          self.paramDictionary["OctavesTerrain"], self.paramDictionary["PersistenceTerrain"]) # Write Octave Noise values to tile array

# Threaded Terrain Generation
    def GenerateThreadedParent(self): # Generates terrain using 4 threads
        threads = []

        halfMap = int(self.MAP_SIZE / 2)
        fullMap = self.MAP_SIZE

        # Create 4 threads for threaded child functions
        threads.append(threading.Thread(target=self.ThreadedChild, args=(0, halfMap, 0, halfMap)))
        threads.append(threading.Thread(target=self.ThreadedChild, args=(halfMap, fullMap, 0, halfMap)))
        threads.append(threading.Thread(target=self.ThreadedChild, args=(0, halfMap, halfMap, fullMap)))
        threads.append(threading.Thread(target=self.ThreadedChild, args=(halfMap, fullMap, halfMap, fullMap)))


        # Start all the threads
        for t in threads:
            t.start()

        # While threads arent finished, pause
        while threading.activeCount() > 1:
            pass

        self.RenderMap() # Render Map
```

```python

    def ThreadedChild(self, x1, x2, y1, y2): # Child Method to GenerateThreadedParent
        for y in range(y1, y2):
            for x in range(x1, x2):
                xCoord = (x / self.MAP_SIZE) * self.paramDictionary["WorldScale"]
                yCoord = (y / self.MAP_SIZE) * self.paramDictionary["WorldScale"]

                self.tileArray[x][y].tileHeight = perlinNoise.OctaveNoise(self.MAP_SEED + xCoord + self.time, self.MAP_SEED + yCoord + self.time,
                                                                          self.paramDictionary["OctavesTerrain"], self.paramDictionary["PersistenceTerrain"]) # Write Octave Noise values to tile array

# Generate Tree Methods
    def GenerateTreeArea(self): # Uses perlin noise to generate the areas for trees to spawn in
        TSO = self.paramDictionary["TreeSeedOffset"]

        treeList = []

        for y in range(0, self.MAP_SIZE):
            for x in range(0, self.MAP_SIZE):
                xCoord = x / self.MAP_SIZE
                yCoord = y / self.MAP_SIZE

                temp = perlinNoise.OctaveNoise(self.MAP_SEED + xCoord + TSO, self.MAP_SEED + yCoord + TSO,
                            self.paramDictionary["OctavesTrees"], self.paramDictionary["PersistenceTrees"]) # Sample octave noise

                tileValue = self.Clamp(((self.tileArray[x][y].tileHeight / 2) + 0.5), 0.0, 1.0) # Clamp value

                if (temp > self.paramDictionary["TreeHeight"] and tileValue > self.paramDictionary["Coast"] + self.paramDictionary["TreeBeachOffset"] and
                                                                          tileValue < self.paramDictionary["Grass"] - self.paramDictionary["TreeBeachOffset"]): # Check within range
                    treeList.append([x, y])

        poissonArray = self.PoissonDiscSampling(treeList) # Get Poisson Disc Sampling values for poisson array

        for y in range(0, self.MAP_SIZE):
            for x in range(0, self.MAP_SIZE):
                self.tileArray[x][y].ClearObject() # Clear Existing objects from tile map

                if poissonArray[x][y] == True:
                    self.tileArray[x][y].AddObject(self.paramDictionary["TreeType"], self.paramDictionary["ColourTree"]) # Add Poisson Disc Sample results to tile map

```

```
    def PoissonDiscSampling(self, pointList): # A tweaked version of poisson disc sampling in 2
    ↪ dimensions
        k = self.paramDictionary["PoissonKVal"]

        pickedPoints = [[False for i in range(self.MAP_SIZE)] for j in range(self.MAP_SIZE)] # Blank
        ↪ array of False

        numPoints = len(pointList) - 1
        if numPoints <= 0: # Catches if no points
            return pickedPoints

        sampleNum = 0

        while sampleNum <= k: # While sampled attempts is less than k
            sample = pointList[random.randint(0, numPoints)]

            result = self.PoissonCheckPoint(sample, pickedPoints) # Check points
            if result == True:
                pickedPoints[sample[0]][sample[1]] = True
                sampleNum = 0
                continue
            else:
                sampleNum += 1
                continue

        return pickedPoints

    def PoissonCheckPoint(self, point, pickedPoints): # Checks Specific points around a point for
    ↪ objects
        if (1 <= point[0] and point[0] <= self.paramDictionary["WorldSize"] - 2 and
                  1 <= point[1] and point[1] <= self.paramDictionary["WorldSize"] - 2):
            if pickedPoints[point[0]][point[1] - 1] == True: return False
            elif pickedPoints[point[0] + 1][point[1]] == True: return False
            elif pickedPoints[point[0]][point[1] + 1] == True: return False
            elif pickedPoints[point[0] - 1][point[1]] == True: return False
            elif pickedPoints[point[0]][point[1]] == True: return False
            else: return True

# Render Methods
    def RenderMap(self): # Renders terrain onto Pygame surface
        resolution = self.MAP_SIZE * self.TILE_WIDTH
        self.RenderedMap = pygame.Surface((resolution, resolution))
        self.RenderedMap.set_colorkey((0,0,0))

        if self.paramDictionary["Grayscale"] == 1: # Renders in grayscale if specified
            for y in range(0, self.MAP_SIZE):
                for x in range(0, self.MAP_SIZE):
                    value = self.tileArray[x][y].tileHeight
                    value = (value / 2) + 0.5
                    value = self.Clamp(value, 0.0, 1.0)

                    pygame.draw.rect(self.RenderedMap, (255 * value, 255 * value, 255 * value), ((x *
                    ↪ self.TILE_WIDTH + self.TILE_BORDER),
                            (y * self.TILE_WIDTH + self.TILE_BORDER), self.TILE_WIDTH -
                            ↪ (self.TILE_BORDER * 2), self.TILE_WIDTH - (self.TILE_BORDER * 2)))
```

```
        else:                                          # Else renders in Colour
            for y in range(0, self.MAP_SIZE):
                for x in range(0, self.MAP_SIZE):
                    value = self.tileArray[x][y].tileHeight
                    value = (value / 2) + 0.5
                    value = self.Clamp(value, 0.0, 1.0) # Clamps value between 0 and 1

                    colour = None

                    if value == 0: # Colour ramp for all available colours
                        colour = (0,0,0)
                    elif value < self.paramDictionary["Water"]:
                        colour = tuple(self.paramDictionary["ColourWater"])
                        self.tileArray[x][y].tileType = 0
                        self.tileArray[x][y].tileColour = colour
                    elif value < self.paramDictionary["Coast"]:
                        colour = tuple(self.paramDictionary["ColourCoast"])
                        self.tileArray[x][y].tileType = 1
                        self.tileArray[x][y].tileColour = colour
                    elif value < self.paramDictionary["Grass"]:
                        colour = tuple(self.paramDictionary["ColourGrass"])
                        self.tileArray[x][y].tileType = 2
                        self.tileArray[x][y].tileColour = colour
                    elif value < self.paramDictionary["Mountain"]:
                        colour = tuple(self.paramDictionary["ColourMountain"])
                        self.tileArray[x][y].tileType = 3
                        self.tileArray[x][y].tileColour = colour

                    # Draws correct colour pixel to rendered map - takes into account width and
                    ↪ border
                    pygame.draw.rect(self.RenderedMap, colour, ((x * self.TILE_WIDTH +
                    ↪ self.TILE_BORDER),
                            (y * self.TILE_WIDTH + self.TILE_BORDER), self.TILE_WIDTH -
                            ↪ (self.TILE_BORDER * 2), self.TILE_WIDTH - (self.TILE_BORDER * 2)))

    def RenderInteractables(self): # Renders interactables onto pygame surface
        resolution = self.MAP_SIZE * self.TILE_WIDTH
        self.RenderedInteractables = pygame.Surface((resolution, resolution))
        self.RenderedInteractables.set_colorkey((0,0,0))

        ITB = self.paramDictionary["InteractableTileBorder"]

        for y in range(0, self.MAP_SIZE): # Draw interactables to rendered image
            for x in range(0, self.MAP_SIZE):
                if self.tileArray[x][y].hasObject == True:
                    tile = self.tileArray[x][y]
                    pygame.draw.rect(self.RenderedInteractables, tile.objectColour, ((x *
                    ↪ self.TILE_WIDTH + ITB),
                            (y * self.TILE_WIDTH + ITB), self.TILE_WIDTH - (ITB * 2), self.TILE_WIDTH
                            ↪ - (ITB * 2)))

    def DrawMap(self, window): # Blits the rendered frames onto the passed through window
        window.blit(self.RenderedMap, (0,0))
        self.RenderInteractables()
```

```
        window.blit(self.RenderedInteractables, (0,0))

# Miscellaneous Methods
    def Clamp(self, val, low, high): # Simple function to clamp a value between two numbers - Used to
    ↪ make sure number doesnt go out of bounds
        return low if val < low else high if val > high else val
```

5.6 perlinNoise.py

```
import random, math

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

def OctaveNoise(x, y, octaves, persistence): # Sums multiple levels of perlin noise
    total = 0
    frequency = 1
    amplitude = 1
    maxValue = 0

    for i in range(octaves): # Combines Multiple octaves of perlin noise
        total += ((Noise(x * frequency, y * frequency)) * amplitude)

        maxValue += amplitude

        amplitude *= persistence
        frequency *= 2

    return total / maxValue

def Noise(x, y): # Returns a value of the perlin noise function at (x, y) coordinate
    xi = math.floor(x) % 255
    yi = math.floor(y) % 255

    g1 = p[p[xi] + yi]
    g2 = p[p[xi + 1] + yi]
    g3 = p[p[xi] + yi + 1]
    g4 = p[p[xi + 1] + yi + 1]

    xf = x - math.floor(x)
    yf = y - math.floor(y)

    d1 = Grad(g1, xf, yf)
```

```python
    d2 = Grad(g2, xf - 1, yf)
    d3 = Grad(g3, xf, yf - 1)
    d4 = Grad(g4, xf - 1, yf - 1)

    u = Fade(xf)
    v = Fade(yf)

    x1Inter = Lerp(u, d1, d2)
    x2Inter = Lerp(u, d3, d4)
    yInter = Lerp(v, x1Inter, x2Inter)

    return yInter

def Grad(hash, x, y): # Gradient Function defined as part of the algorithm
    temp = hash & 3
    if temp == 0:
        return x + y
    elif temp == 1:
        return -x + y
    elif temp == 2:
        return x - y
    elif temp == 3:
        return -x - y
    else:
        return 0

def Lerp(ammount, left, right): # Linear interpolation of values
    return ((1 - ammount) * left + ammount * right)

def Fade(t): # Fade Function defined as part of the algorithm
    return t * t * t * (t * (t * 6 - 15) + 10)
```

5.7 deepqlearning.py

```python
from audioop import bias
import random, pickle, math
from typing import final
from matrix import Matrix
import activations
from copy import copy
from datalogger import *
import time

class DoubleNeuralNet(): # Wraps a Main and Target Neural Network together
    def __init__(self, layers, params, load=False, loadName="DQNetwork"): # Constructor for a Double
    ↪ Neural Network
        self.paramDictionary = params

        if not load: # Create brand new values
            self.MainNetwork = NeuralNet(layers, params)
            self.TargetNetwork = NeuralNet(layers, params)

            self.ExperienceReplay = Deque(self.paramDictionary["ERBuffer"])

            self.epsilon = self.paramDictionary["DQLEpsilon"]
```

```python

            self.step = 0
            self.cumReward = 0.0

            self.layerActivation = activations.Sigmoid()
            self.finalLayerActivation = activations.SoftMax()
        else:
            self.LoadState(loadName) # Load values from saved data

        self.fileName = loadName

        self.activations = (self.layerActivation, self.finalLayerActivation) # Tuple of activations

        self.batchReward = 0
        self.maxBatchReward = 0
        self.batchLoss = 0
        self.dataPoints = []

                                                        # BatchReward, MaxBatchReward, PercentageDifference, Step
        self.actionTracker = DataLogger("ActionTracker", [[float, int], [float, int], [float, int], int], False)

        self.startTime = time.time()

    def TakeStep(self, agent, worldMap, enemyList): # Takes a step forward in time
        self.step += 1

        # Forward Propagation
        agentSurround = agent.GetTileVector(worldMap, enemyList)
        postProcessedSurround = agent.TileVectorPostProcess(agentSurround) # Retrieve Vector of State info from Agent
        netInput = postProcessedSurround[1]

        self.MainNetwork.ForwardPropagation(netInput, self.activations) # Forward Prop the Main Network

        output = self.MainNetwork.layers[-1].activations
        outputMax = output.MaxInVector()

        # Action Taking and Reward
        if random.random() > self.epsilon:
            softmaxxed = self.finalLayerActivation.Activation(copy(output))
            action = random.randint(0, self.paramDictionary["DeepQLearningLayers"][-1] - 1)
            val = random.random()
            totalled = 0
            for i in range(softmaxxed.order[0]):
                totalled += softmaxxed.matrixVals[i][0]
                if totalled >= val:
                    action = i
                    break
        else:
            action = random.randint(0, self.paramDictionary["DeepQLearningLayers"][-1] - 1)

```

```
        rewardVector = agent.GetRewardVector(agentSurround, self.paramDictionary["DeepQLearningLayers"][-1])
        reward = rewardVector.matrixVals[action][0] # Get reward given action
        self.cumReward += reward
        self.batchReward += reward
        self.maxBatchReward += rewardVector.MaxInVector()[0]

        agent.CommitAction(action, agentSurround, worldMap, enemyList) # Take Action
        # Epsilon Regression
        self.epsilon *= self.paramDictionary["DQLEpisonRegression"]

        # Assigning values to tempExperience
        tempExp = Experience()
        tempExp.state = agentSurround
        tempExp.action = action
        tempExp.reward = rewardVector
        tempExp.stateNew = agent.GetTileVector(worldMap, enemyList)

        self.ExperienceReplay.PushFront(copy(tempExp))

        # Back Propagation
        expectedValues = self.ExpectedValue(output, tempExp, agent) # Calculating Loss

        Cost = self.HalfSquareDiff(output, expectedValues)

        self.batchLoss += Cost.Sum()

        self.MainNetwork.layers[-1].errSignal = Cost * self.layerActivation.Derivative(copy(self.MainNetwork.layers[-1].preactivations))

        self.MainNetwork.BackPropagationV2(self.activations) # Back Propagating the loss

        # Do things every X steps passed
        if self.step % self.paramDictionary["TargetReplaceRate"] == 0: # Replace Weights in Target Network
            self.TargetNetwork.layers = self.MainNetwork.layers

        # Sample Experience Replay Buffer
        if (self.paramDictionary["EREnabled"] and self.step % self.paramDictionary["ERSampleRate"] == 0 and self.ExperienceReplay.Full()):
            self.SampleExperienceReplay(agent)

        # Actions to run after every Batch
        if self.step % self.paramDictionary["DQLEpoch"] == 0:
            print(self.step, self.cumReward, self.epsilon, time.time() - self.startTime)

            self.MainNetwork.UpdateWeightsAndBiases(self.paramDictionary["DQLEpoch"]) # Update weights and biases

            if self.paramDictionary["SaveWeights"]: # Saves weights if specified
                self.SaveState(self.fileName)

            #Log Action
            self.actionTracker.LogDataPoint([self.batchReward, self.maxBatchReward, self.batchLoss, self.step])
```

```
                #self.actionTracker.LogDataPointBatch(self.dataPoints)

                self.dataPoints = []
                self.actionTracker.SaveDataPoints()

                self.batchReward = 0
                self.maxBatchReward = 0
                self.batchLoss = 0

    def SampleExperienceReplay(self, agent): # Samples the Experience Replay Buffer, Back Propagating
    ↪   its Findings
        print("Sampling Experience Replay")
        samples = self.ExperienceReplay.Sample(self.paramDictionary["ERSampleSize"])

        for sample in samples:
            postProcessedSurround = agent.TileVectorPostProcess(sample.state) # Post process the Tile
            ↪   Vector
            netInput = postProcessedSurround[1]

            self.MainNetwork.ForwardPropagation(netInput, self.activations) # Forward Prop the Main
            ↪   Network

            output = self.MainNetwork.layers[-1].activations

            expectedValues = self.ExpectedValue(output, sample, agent) # Calculating Loss

            Cost = self.HalfSquareDiff(output, expectedValues)

            self.MainNetwork.layers[-1].errSignal = Cost *
            ↪   self.layerActivation.Derivative(copy(self.MainNetwork.layers[-1].preactivations))

            self.MainNetwork.BackPropagationV2(self.activations) # Back Propagating the loss

    def HalfSquareDiff(self, networkOutput, expected):
        return ((expected - networkOutput) ** 2) * 0.5

    def ExpectedValue(self, output, tempExp, agent):
        # L^i(W^i) = ((r + y*maxQ(s',a';W^i-1) - Q(s,a,W)) ** 2
        # Loss = ((Reward[] + Gamma * MaxQ(s', a'; TNet)) - Q(s, a)[]) ^ 2

        Reward = tempExp.reward
        Gamma = self.paramDictionary["DQLGamma"]

        #self.TargetNetwork.ForwardPropagation(agent.TileVectorPostProcess(tempExp.state)[1],
        ↪   self.activations) # Apply input to Target Network

        #targetNetAction = self.TargetNetwork.layers[-1].activations.MaxInVector()[1]


        tempRewardVec = agent.GetRewardVector(tempExp.stateNew,
        ↪   self.paramDictionary["DeepQLearningLayers"][-1]) # Gets reward vector from the new state
        maxQTNet = agent.MaxQ(tempRewardVec) # Max of Target network

        LossVec = ((Reward + (Gamma * maxQTNet)) - output) ** 2 # Bellman Equation
        return LossVec
```

```python

    def SaveState(self, file):
        state = [self.MainNetwork, self.TargetNetwork, self.ExperienceReplay, self.step,
                    self.epsilon, self.cumReward, self.layerActivation, self.finalLayerActivation]
        with open("DQLearningData\\" + file + ".dqn", "wb") as f:
            pickle.dump(state, f)

    def LoadState(self, file): # Returns stored Neural Network data
        with open("DQLearningData\\" + file + ".dqn", "rb") as f:
            state = pickle.load(f)

            self.MainNetwork = state[0]
            self.TargetNetwork = state[1]
            self.ExperienceReplay = state[2]
            self.step = state[3]
            self.epsilon = state[4]
            self.cumReward = state[5]
            self.layerActivation = state[6]
            self.finalLayerActivation = state[7]

class NeuralNet(): # Neural Network Implementation
    def __init__(self, layersIn, params): # Constructor for a Single Neural Network
        self.paramDictionary = params

        newLayersIn = copy(layersIn)

        newLayersIn.append(1)

        self.layers = []

        for i in range(len(newLayersIn) - 1):
            print(newLayersIn[i])
            self.layers.append(Layer(newLayersIn[i], newLayersIn[i + 1]))

    def ForwardPropagation(self, inputVector, activations): # Iterates through Forward Propagation
        self.layers[0].activations = inputVector

        for i in range(0, len(self.layers) - 1):
            self.layers[i].ForwardPropagation(self.layers[i+1], activations)

        #self.layers[-1].ForwardPropagation(self.layers[-2], activations, finalLayer=True)

    def BackPropagationV2(self, activations): # Iterates through Back Propagation V2
        self.layers[-2].BackPropagationV2(self.layers[-1], self.paramDictionary["DQLLearningRate"],
        ↪   activations)

        for i in range(len(self.layers) - 3, 0, -1):
            self.layers[i].BackPropagationV2(self.layers[i+1],
            ↪   self.paramDictionary["DQLLearningRate"], activations)

    def UpdateWeightsAndBiases(self, epochCount): # Update Weights and biases
        for i in range(1, len(self.layers)):
            self.layers[i].UpdateWeightsAndBiases(epochCount)

class Layer(): # Layer for a Neural Network
```

```
    def __init__(self, size, nextSize, inputLayer=False): # Constructor for a Layer Object
        if inputLayer == False: # Additional objects if not the input layer
            pass

        self.weightMatrix = Matrix((nextSize, size), random=True)
        self.biasVector = Matrix((nextSize, 1), random=False)

        self.weightUpdates = Matrix((nextSize, size))
        self.biasUpdates = Matrix((nextSize, 1))

        self.errSignal = Matrix((nextSize, 1))
        self.preactivations = Matrix((size, 1))
        self.activations = Matrix((size, 1))

    def ForwardPropagation(self, nextLayer, activations): # Forward Propagates the Neural Network
        self.preactivations = self.weightMatrix * self.activations + self.biasVector

        nextLayer.activations = activations[0].Activation(copy(self.preactivations))

    def BackPropagationV2(self, prevLayer, lr, layerActivations): # 2nd Revision of Back Propagation
        deltaWeightProduct = (prevLayer.weightMatrix.Transpose() * prevLayer.errSignal)
        self.errSignal = deltaWeightProduct *
        ↪ layerActivations[0].Derivative(copy(self.preactivations))

        weightDerivatives = self.errSignal * self.activations.Transpose()
        biasDerivatives = self.errSignal

        self.weightUpdates += weightDerivatives * lr
        self.biasUpdates += biasDerivatives * lr

    def UpdateWeightsAndBiases(self, epochCount): # Update Weights and Biases
        self.weightMatrix -= (self.weightUpdates * (1 / epochCount))
        self.biasVector -= (self.biasUpdates * (1 / epochCount))

        self.weightUpdates.Clear()
        self.biasUpdates.Clear()

class Experience(): # Used in Experience Replay
    def __init__(self, state = None, action = None, reward = None, stateNew = None): # Constructor
    ↪ for an Experience
        self.state = state
        self.action = action
        self.reward = reward
        self.stateNew = stateNew

class Deque(): # Partial Double Ended Queue Implementation
    def __init__(self, length):
        self.length = length

        self.queue = [None for i in range(self.length)]

        self.frontP = -1
        self.backP = -1

    def PushFront(self, item): # Pushes item to front of Queue
```

```
        self.frontP = (self.frontP + 1) % self.length

        if self.queue[self.frontP] != None:
            self.backP = (self.frontP + 1) % self.length

        self.queue[self.frontP] = item

    def Full(self): # Checks if Queue is full
        if self.queue[self.length - 1] != None:
            return True
        return False

    def First(self): # Returns Front Item from Queue
        return self.queue[self.frontP]

    def Last(self): # Returns Final Item from Queue
        return self.queue[(self.frontP + 1) % self.length]

    def Sample(self, n): # Samples N number of samples from the deque
        temp = self.queue
        return random.sample(temp, n)
```

5.8 activations.py

```
from abc import ABC, abstractmethod
from math import e, tanh, exp, cosh
from matrix import *

class Activation(ABC): # Abstract Base Class
    @abstractmethod
    def Activation(self, x): # Abstract Activation Method
        pass

    @abstractmethod
    def Derivative(self, x): # Abstract Derivative Method
        pass

class ReLu(Activation): # ReLu
    def __init__(self):
        pass

    def Activation(self, x): # Returns value if greater than 0, else 0
        for row in range(x.order[0]):
            x.matrixVals[row][0] = max(0, x.matrixVals[row][0])
        return x

    def Derivative(self, x): # If value is greater than 0 return 1, else return 0
        for row in range(x.order[0]):
            if x.matrixVals[row][0] >= 0: x.matrixVals[row][0] = 1
            else: x.matrixVals[row][0] = 0
        return x

class LeakyReLu(Activation): # Leaky ReLu
    def __init__(self):
        pass
```

```

    def Activation(self, x): # Returns value if greater than 0, else a apply a gradient to x and
    ↪   return it
        for row in range(x.order[0]):
            x.matrixVals[row][0] = max(x.matrixVals[row][0] * 0.1, x.matrixVals[row][0])
        return x

    def Derivative(self, x): # If value is greater than 0 return 1, else return 0.01
        for row in range(x.order[0]):
            if x.matrixVals[row][0] >= 0: x.matrixVals[row][0] = 1
            else: x.matrixVals[row][0] = 0.1
        return x

class Sigmoid(Activation): # Sigmoid
    def __init__(self):
        pass

    def Activation(self, x): # Mathematical Function to get "squish" values between 0 and 1
        for row in range(x.order[0]):
            if x.matrixVals[row][0] > 15: x.matrixVals[row][0] = 1
            elif x.matrixVals[row][0] < -15: x.matrixVals[row][0] = 0
            else: x.matrixVals[row][0] = 1 / (1 + exp(-x.matrixVals[row][0]))
        return x

    def Derivative(self, x): # Derivative of the Sigmoid Function
        for row in range(x.order[0]):
            sigmoidSingle = self.ActivationSingle(x.matrixVals[row][0])
            x.matrixVals[row][0] = sigmoidSingle * (1 - sigmoidSingle)
        return x

    def ActivationSingle(self, x): # Single value for use in the derivative
        if x > 15: return 1
        elif x < -15: return 0
        else: return 1 / (1 + exp(-x))

class SoftMax(Activation): # SoftMax
    def __init__(self):
        pass

    def Activation(self, x): # Returns a probability distribution between a vector of values
    ↪   totalling to 1
        sumToK = 0

        for i in range(x.order[0]):
            sumToK += exp(x.matrixVals[i][0])

        outVector = Matrix(x.order)

        for i in range(x.order[0]):
            outVector.matrixVals[i][0] = (exp(x.matrixVals[i][0])) / sumToK

        return outVector # Returns vector and best index

    def Derivative(self, x): # Derivative of the softmax function
        for row in range(x.order[0]):
```

```
                x.matrixVals[row][0] = x.matrixVals[row][0] * (1 - x.matrixVals[row][0])

        return x

class NullActivation(Activation): # No activation function
    def __init__(self):
        pass

    def Activation(self, x): # Returns the same values
        return x

    def Derivative(self, x): # Returns the same values
        return 1

class TanH(Activation): # TanH
    def __init__(self):
        pass

    def Activation(self, x): # TanH mathematical function
        for row in range(x.order[0]):
            x.matrixVals[row][0] = tanh(x.matrixVals[row][0])
        return x

    def Derivative(self, x): # Derivative of TanH
        for row in range(x.order[0]):
            x.matrixVals[row][0] = (1 / (cosh(x.matrixVals[row][0]))) ** 2
        return x
```

5.9 datalogger.py

```
import pickle, random
from heap import *

# Data Logger Class for logging information for analysis
class DataLogger():
    def __init__(self, name, dataStructure, load=True): # Constructor Method
        self.name = name

        self.dataStructure = dataStructure

        if load: # Loads Data if available but else create blank
            self.dataPoints = DataLogger.LoadDataPoints(name)
        else:
            self.dataPoints = []

    def LogDataPointBatch(self, dataPoints): # Logs a Batch of Data Points
        for i in range(len(dataPoints)):
            self.LogDataPoint(dataPoints[i])

    def LogDataPoint(self, dataPoint): # Logs Data Point to Data Point list
        if self.CheckMatchStructure(dataPoint):
            self.dataPoints.append(dataPoint)


    def CheckMatchStructure(self, dataPoint): # Checks the given Data Point is in the correct Form
```

```python
        if len(dataPoint) != len(self.dataStructure): # Throws error if lengths dont match
            raise Exception("Structure of Data Point does not match Collector Specified Structure")

        for i in range(len(dataPoint)):
            t1 = type(dataPoint[i]) # Type 1
            t2 = self.dataStructure[i] # Type 2

            if t1 == list and type(t2) != list: # Checks if list is all of same type
                flag = False

                for x in range(len(dataPoint[i])):
                    if type(dataPoint[i][x]) != t2:
                        flag = True
                if not flag:
                    continue

            elif t1 == list and type(t2) == list: # Checks list against list
                if len(dataPoint[i]) == len(t2):
                    flag = False
                    for x in range(len(dataPoint[i])):
                        if type(dataPoint[i][x]) != t2[x]:
                            flag = True

                    if not flag:
                        continue

            elif type(t2) == list: # Checks Multiple types against t1
                flag = False

                for x in range(len(t2)):
                    if t1 == t2[x]:
                        flag = True
                if flag:
                    continue

            else:                 # Checks Singular type against t1
                if t1 == t2:
                    continue

            raise Exception(("Type: {} != Data Structure Type: {} \n {}").format(t1, t2, self.dataStructure))
        return True

    def HeapSort(self, parameterIndex): # O(n*log n) sorting algorithm utilising a Heap Data structure, Sorts the data points by the specified parameter
        # 1000 Items -> 0.13
        # 10000 Items -> 12.1
        # 100000 Items -> 1646 or 27.4 minutes

        if type(self.dataStructure[parameterIndex]) == list: # Throw error if data structure element is List
            raise Exception("Cannot sort by structure: {}".format(type(self.dataStructure[parameterIndex])))

```

```python
        elif self.dataStructure[parameterIndex] == bool: # Throw error if data structure element is
        ↪ Bool
            raise Exception("Cannot sort by structure:
            ↪ {}".format(self.dataStructure[parameterIndex]))

        sortedList = []

        heap = Heap(self.dataPoints, parameterIndex) # Creates a new heap

        while heap.Length() - 1 >= 0:
            sortedList.append(heap.RemoveTop()) # Loops popping and appending greatest element from
            ↪ Heap

        return sortedList

    def Select(self, searchIndex, searchContents): # Select a specified element with contents from
    ↪ data points
        returnedList = []

        for i in range(len(self.dataPoints)):
            if self.dataPoints[i][searchIndex] in searchContents:
                returnedList.append(self.dataPoints[i])

        return returnedList

    # Using Pickle to Save/Load
    @staticmethod
    def LoadDataPoints(file): # Returns stored dataPoints
        with open("DataLogger\\" + file + ".data", "rb") as f:
            temp = pickle.load(f)
        return temp

    def SaveDataPoints(self): # Saves dataPoints to a file
        with open("DataLogger\\" + self.name + ".data", "wb") as f:
            pickle.dump(self.dataPoints, f)
```

5.10 heap.py

```python
import math

# A Binary tree with the heap property, such that for every element, both children are <= to the
↪ parent
class Heap:
    def __init__(self, elements, indexIn): # Creates a new heap from a list of elements, and assigns
    ↪ an index for which to sort by
        self.elements = elements
        self.index = indexIn

        self.Heapify()

    def AddElement(self, element): # Adds Singular element to Heap
        self.elements.append(element)
        self.SiftUp(len(self.elements) - 1)

    def SiftUp(self, elementIndex): # Sifts a singular element up the heap if possible
```

```python
        newElementIndex = elementIndex
        isHeap = False

        while not isHeap: # Repeat until is a heap again
            parentIndex = math.floor((newElementIndex - 1) / 2)

            if parentIndex == 0 and newElementIndex == 0: # Base Case
                isHeap = True

            elif self.elements[newElementIndex][self.index] >= self.elements[parentIndex][self.index]: # Swaps elements which dont conform to heap property
                tempSwap = self.elements[parentIndex]
                self.elements[parentIndex] = self.elements[newElementIndex]
                self.elements[newElementIndex] = tempSwap

                newElementIndex = parentIndex
            else:
                isHeap = True

    def SiftDown(self, elementIndex): # Sifts a singular element down the heap if possible
        rootIndex = elementIndex
        isHeap = False

        end = len(self.elements) - 1

        while ((2 * rootIndex) + 1) <= end: # Repeat until the next root index is outside the dimensions of the heap
            childIndex = (rootIndex * 2) + 1

            if childIndex + 1 <= end and self.elements[childIndex][self.index] < self.elements[childIndex + 1][self.index]: # Checks which child is larger
                childIndex += 1

            if self.elements[rootIndex][self.index] < self.elements[childIndex][self.index]: # Swapping elements which dont conform to Heap rules
                tempSwap = self.elements[childIndex]
                self.elements[childIndex] = self.elements[rootIndex]
                self.elements[rootIndex] = tempSwap

                rootIndex = childIndex
            else:
                break

    def RemoveTop(self): # Pops top element off of Heap and returns it, heapifies the heap once removed
        tempSwap = self.elements[-1]
        self.elements[-1] = self.elements[0] # Swaps First and Last elements
        self.elements[0] = tempSwap

        returnElement = self.elements[-1] # Stores and deletes the final element
        self.elements = self.elements[:-1]

        self.Heapify() # Creates Heap again

```

```python
        return returnElement # Returns Top element

    def Peek(self): # Returns root/top element
        return self.elements[0]

    def Length(self): # Returns size of heap
        return len(self.elements)

    def Heapify(self): # Returns values to a heap form, where all children of parents are less than
    ↪ or equal too
        for i in range(math.floor((len(self.elements) - 1) / 2), -1, -1):
            self.SiftDown(i)
```

5.11 plotData.py

```python
import matplotlib.pyplot as plt
import pickle
from os import listdir
from os.path import isfile, join
from typing import DefaultDict

def LoadFileList(dir): # Locating files in directory and returning them as a dictionary
    directoryList = listdir(dir)
    validFiles = [f for f in directoryList if isfile(join(dir, f))]

    fileDict = DefaultDict(str)

    for i in range(len(validFiles)):
        fileDict[i] = validFiles[i]

    return fileDict

def PickChoice(fileDict): # Pick choice from file dictionary
    print("List of Data Files:")
    for file in fileDict:
        print(str(file) + " : " + fileDict[file])

    inp = eval(input())
    if isinstance(inp, int):
        return fileDict[inp]
    else:
        raise Exception("Not a valid input")

def LoadPoints(file): # Load Data Points from file
    dataPoints = []
    with open("DataLogger\\" + file, "rb") as f:
        dataPoints = pickle.load(f)
    return dataPoints

# Logic
fileDictionary = LoadFileList("DataLogger\\")
file = PickChoice(fileDictionary)
dataPoints = LoadPoints(file)

print("Plot: ")
```

```python
inp = eval(input())

plottedData = [dataPoints[i][inp] / 100 for i in range(len(dataPoints))]
step = [dataPoints[i][-1] for i in range(len(dataPoints))]

# Setup Plot
plt.plot(step, plottedData)
plt.xlabel("Step Count")
plt.ylabel("Average Loss per Step")

plt.show()
```

5.12 Parameters File

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

    "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": 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" : [25, 32, 16, 8, 6],
    "DQLEpoch": 100,
    "DQLearningMaxSteps": 500000,
    "DQLOffset": 2,
    "DQLEpsilon": 0.5,
    "DQLEpisonRegression": 0.99998,
    "DQLLearningRate": 0.75,
    "DQLGamma": 0.8
}
```

5.13 Ranges File

```
{
    "EnterValues": [0, 1],
    "GenerateThreaded": [0, 1],
    "EnableEnemies": [0, 1],
    "SaveWeights": [0, 1],
    "StepDelay": [0, null],
    "Debug": [0, 1],
    "DebugScale": [0, 4],

    "WorldSize": [16, 1024],
    "TileWidth": [1, 8],
    "TileBorder": [0, 3],

    "OctavesTerrain": [1, 20],
    "PersistenceTerrain": [0, 1],
    "WorldScale": [0.1, 10],

    "OctavesTrees": [1, 20],
    "PersistenceTrees": [0, 1],
    "PoissonKVal": [0, null],
    "TreeSeedOffset": [0, null],
    "TreeHeight": [0, 1],
    "InteractableTileBorder": [0, 3],
    "TreeBeachOffset": [0, 1],

    "Grayscale": [0, 1],
    "Water": [0, 1],
```

```
    "Coast": [0, 1],
    "Grass": [0, 1],
    "Mountain": [0, 1],

    "StartEnemyCount": [0, null],

    "MoveReward": [-1, 1],
    "CollectItemReward": [-1, 1],
    "DeathReward": [-1, 1],
    "ExploreReward": [-1, 1],
    "AttackReward": [-1, 1],
    "AttackFailedReward": [-1, 1],
    "NoopReward": [-1, 1],

    "TargetReplaceRate": [5, 300],
    "EREnabled": [0, 1],
    "ERBuffer": [1000, 10000],
    "ERSampleRate": [1, 100],
    "ERSampleSize": [10, 1000],

    "DQLEpoch": [10, 1000],
    "DQLearningMaxSteps": [1000, null],
    "DQLOffset": [1, 10],
    "DQLEpsilon": [0, 1],
    "DQLEpisonRegression": [0, 1],
    "DQLLearningRate": [0, 1],
    "DQLGamma": [0, 1]
}
```