An Investigation into Machine Learning and Neural Networks 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 by developing a survival simulation environment in which a character will be controlled by a Machine Learning algorithm. The survival simulation will present multiple challenges such as dynamic threats towards the agent in order to provide a complex problem for it to solve. The key question I aim to answer with this investigation is:

Can you train a Machine Learning algorithm to survive in a pseudo random, open-world environment?

I find this question to be quite interesting because there is multiple layers of complexity to it, with several different problems to solve. Answering the question will require me to dive headfirst into Machine Learning picking things up as fast as possible.

1.2 Background

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.

He has agreed to answer some questions for my Interview once I have completed my Initial Investigation.

1.4 First Interview

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

1. What are your first impressions of my project?

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

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

   "Without pushing your complexity too far I think you should look into Deep Reinforcement Learning, I believe it has the possibility of solving your problem if not too complex. Because of that you may 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 in your Evaluation."

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."

1.5 Initial Research

1.5.1 Existing Investigations

Crafter

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

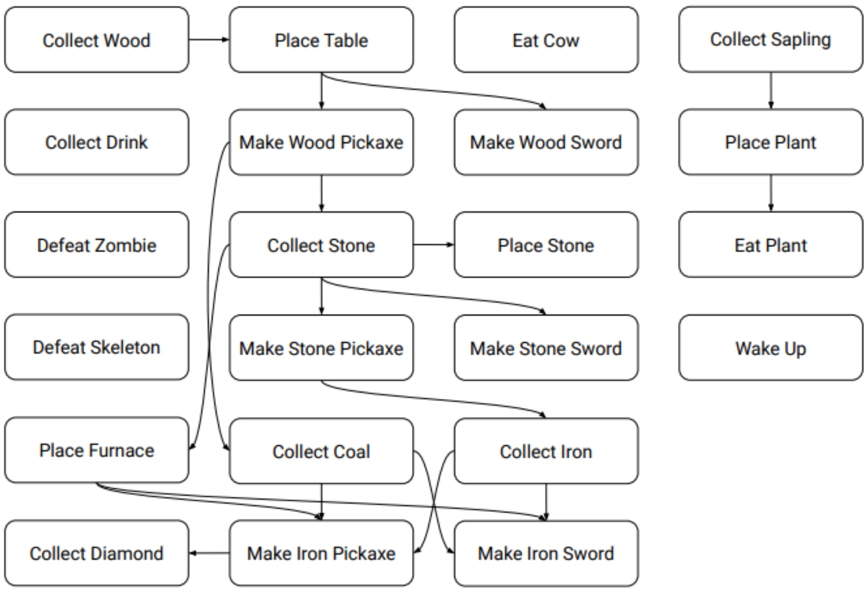
https://github.com/danijar/crafter

Crafter is described to be *"Benchmarking the Spectrum of Agent Capabilities"*, and is utlised in conjunction with Machine Learning Algoriths 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.

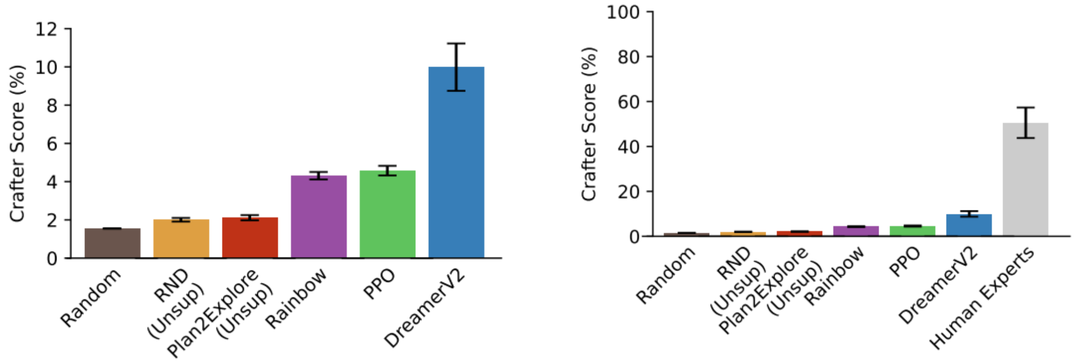
Long-Term reasoning is a complex problem to solve in the context of Machine Learning, current Machine Learning models struggle to deal with this problem. This is dealt with by using algorithms built to mimic "memory". A common implementation of this is Experience Replay which stores states in a queue,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



(a) Success Rate of Algorithms (b) Comparison Against Human Data

Algorithms and Human Experts.

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

Minecraft

Minecraft is a *very* popular Game. It's a sandbox game, meaning that the player can do almost anything they want. The game is formed from blocks which can be broken or placed, along with a plethera of items, enemies, passive animals and more. It has infinite terrain generation, and explicity uses Perlin Noise, and is generated from a seed. The seed 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.

Example of Minecraft's terrain generation in a Swamp Biome

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.

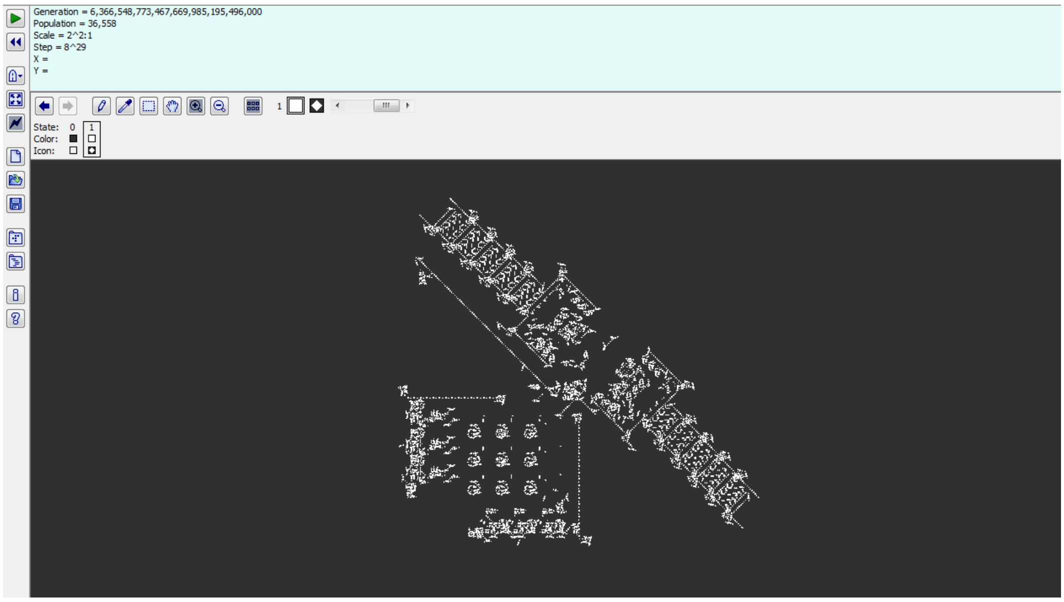
Conway's Game of Life

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:



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.

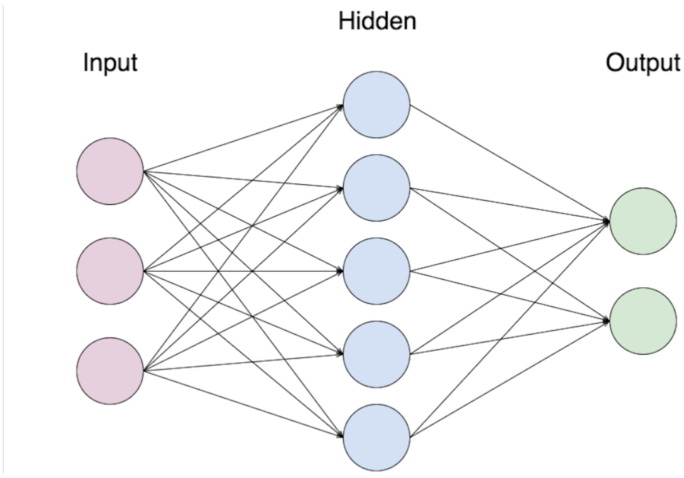
1.6 Algorithms and Potential Data Types

Neural Network and Matrices

As part of developing a Machine Learning Algorithm, I will need to implement a Matrix class in order to implement a neural network. Matrices are commonly used to represent individual layers of a network. Along with making calculations much easier, condensing them into performing operations on matrices, rather than nested 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 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 $[3, 5, 2]$.

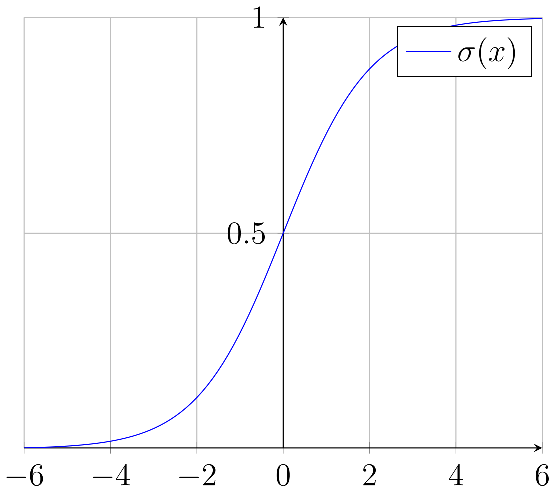


Each connection, otherwise known as an Arc or Edge, has an associated weight. Along with every output of a layer having an associated Bias. These are used to compute the outcome of a network. Forward Propagation is used to compute the outcome of a network, it has a general form and uses Matrix Multiplication 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.

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.

Proposed Programming Language and Associated Libraries

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

Proposed Solution	Benefits and Downsides of Proposed Solution

Python	Python is the first thought which comes to mind when I think about programming, it is my favourite language and I'm yet to find anything which I prefer. Its very versatile and great for rapid prototyping, the dynamic typing makes It great for coding quickly without worrying too much about whether you're using a *float32 or float64*. It also has hundreds of libraries and is very well supported by its developers and the community.	
Python Graphical Libraries	Pygame	Pygame is a highly customizable and well developed binding of *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.
	Tkinter	Tkinter provides an interface to the standard *Tcl/Tk GUI Toolkit*, which is available for most platforms, this makes it highly versatile. Though as my project is not intended as a software package I dont see this as being an incredibly big selling point. Tkinter will serve mostly the same purpose as Pygame but give me easier options for Graphical Input, I dont currently plan to add GUI so this feature isnt neccesary.
C#	C# is my second favourite language, I have plenty of experience with it from developing games with Unity. Its faster than Python and is less abstracted, but this speed isn't necessarily required for my project. With C# I could utilise the *Unity Game Engine* for my project, but then I might end-up relying on builtin types and functions rather than developing my own.	

Proposed Solution	Benefits and Downsides of Proposed Solution	
C# Graphical Libraries	Windows Forms	Windows Forms is a relatively simple drag drop interface for designing your own applications. I've never used it before but I could utilise it with C# to create my project. I belive it might be a bit overkill for my needs though, as it includes many, many UI features which I will have no use for.
	WPF	WPF or *Windows Presentation Foundation* is a versatile development platform for desktop applications. It is relatively versatile in its uses and utilises XAML and is the UI Language of Windows Platforms. XAML would be a new language for me to learn but I have experience with HTML so I dont believe it would be too difficult. The platform would provide a stable base to my project.
Rust	Rust is low level language designed for speed and efficiency, I started using it recently as a side hobby and would like to use it more in future projects of mine. Though I feel like it may be a bit overkill for a Computer Science NEA, with it often being used for server side applications rather than general purpose applications.	
Rust Graphical Libraries	Piston2d	Piston2d is a feature complete 2d graphics library which utilises OpenGl, I've worked with it briefly before and I believe it would be a good option over Pixels if I needed more complex drawing methods.
	Pixels	Pixels is a lightweight 2d graphics library designed to simply push pixels to the screen, Its relatively simple and ive used it for making a simple *Falling Sand Game* before, could be a good little option if I wanted to develop a lightweight solution.

1.7 Prototype

1.7.1 Terrain Generation and Displaying

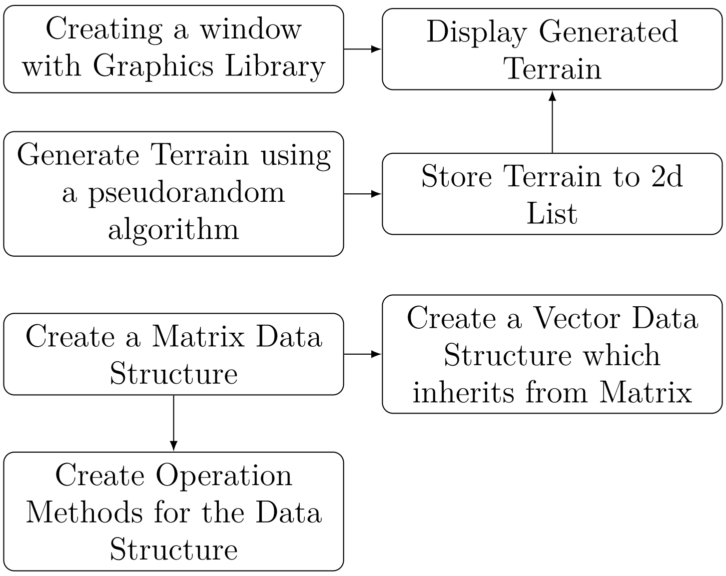
Before starting my Prototype I had to decide upon a short list of objectives I wanted to complete/investigate as part of it. These boiled down to a few things:

1. Terrain Generation
2. Displaying the Generated Terrain using a Graphics Library
3. 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 Interpretted language which makes it versatile for protyping and fast, iterative development.

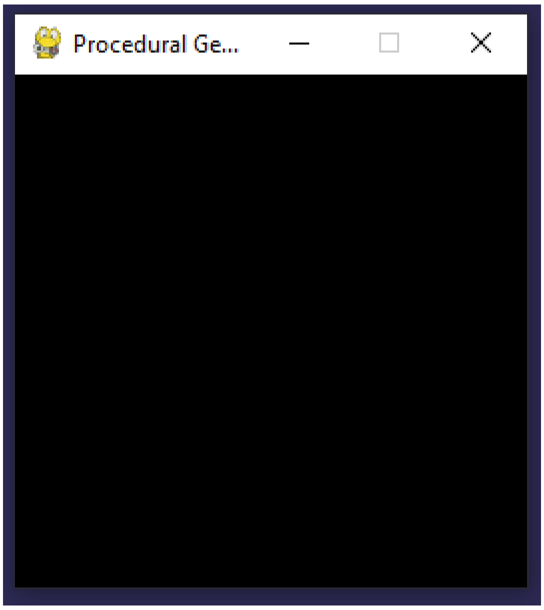
1.7.2 Terrain Generation and Displaying

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:

```
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:

$$Up- > pullValue = upTiles * 0.09$$
$$Down- > pullValue = upTiles * -0.08$$
$$Neutral- > pullValue = 0$$

$$Value[x][y] += pullValue$$

We then add the pullValue to the original square value, leaving us with the updated value. The code for this shown under the Prototype Code Header.

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 grants us the average:

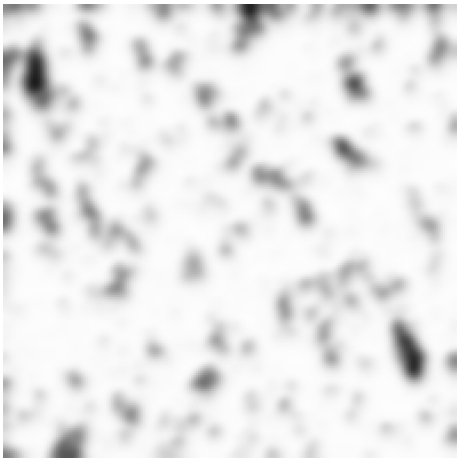
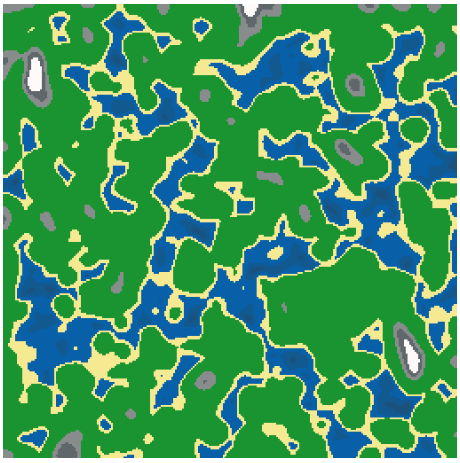
$$\frac{0.83 + 0.93 + 0.64 + 0.07 + 0.38 + 0.21 + 0.95 + 0.33 + 0.94}{9} = 0.586$$

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

The code for this shown under the Prototype Code Header.

1.7.3 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.

1.8 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:

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

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.

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

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 $M3$.

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

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

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

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.

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

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

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.

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} . \begin{pmatrix} d \\ e \\ f \end{pmatrix} = a*d + b*e + c*f$$

All code is available under the Prototype Code Header.

1.8.1 Prototype Evaluation

Overall I am happy with my prototype, though I feel like some parts need to be improved. I did meet my objectives for my prototype but there were improvements which can me made when I create my Technical Solution. Namely the Terrain Generation along with the Matrix class. I feel that Perlin noise would be a better alternative to the two algorithms I used. In theory it should produce better results, and also provice more marks for complexity. My Matrix class could be rewritten to be more efficient, along with using operator overloading, which I didnt know Python could do at the time. I also feel like having vector inherit from matrix is relatively pointless, there is no need for it when I could just use 1 wide Matrices.

1.9 Second Interview

1.

1.10 Objetives

Taking into account my Prototype and Interview, I have formed a list of objectives I feel to be most appropriate for my Investigation. If all completed they will form a complete solution which will answer my Investigations question. Below is the list of objectives split into 6 key sections:

1. User Input
   (a) Read Parameters from a Json formatted file
   (b) Check Parameters fall within a certain range to prevent errors
   (c) Give user option to load Neural Network Training progress
2. Simulation
   (a) Utilise Perlin Noise to generate a 2d List of terrain heights
   (b) Store Terrain Heights in a Tile Data Type
   (c) Utilise Threading to generate Terrain Faster
   (d) Display terrain to a window
   (e) Map ranges of terrain heights to specific colour bands
   (f) Utilise Poisson Disc Sampling to generate objects for the Agent to interact with
   (g) Implement enemies which use basic pathfinding to traverse towards the player
   (h) Generate multiple enemies upon starting the simulation
   (i) Allow the enemies to attack the Agent
3. Agent
   (a) Implement Movement options for the Agent
   (b) Implement the ability to pick up the generated Objects
   (c) Implement the ability to attack the generated enemies
   (d) Create methods to sample the terrain around the Agent
   (e) Create methods to convert the sampled Tiles into a grayscale input vector for a neural network
   (f) Create reward methods to reward the agent given the terrain samples and action
4. Matrix Class
   (a) Implement a Dynamic Matrix Class with appropriate Operations such as:
      i. Multiplication
      ii. Addition
      iii. Subtraction
      iv. Transpose
      v. Sum
      vi. Select Row/Column
   (b) Create appropriate errors to throw when utilising methods the incorrect way
5. Deep Q Learning
   (a) Dynamically create a Dual Neural Network model based upon loaded parameters
   (b) Implement an Abstract Class for Activation Functions
   (c) Implement Activation Functions inheriting from the Abstract Class such as:

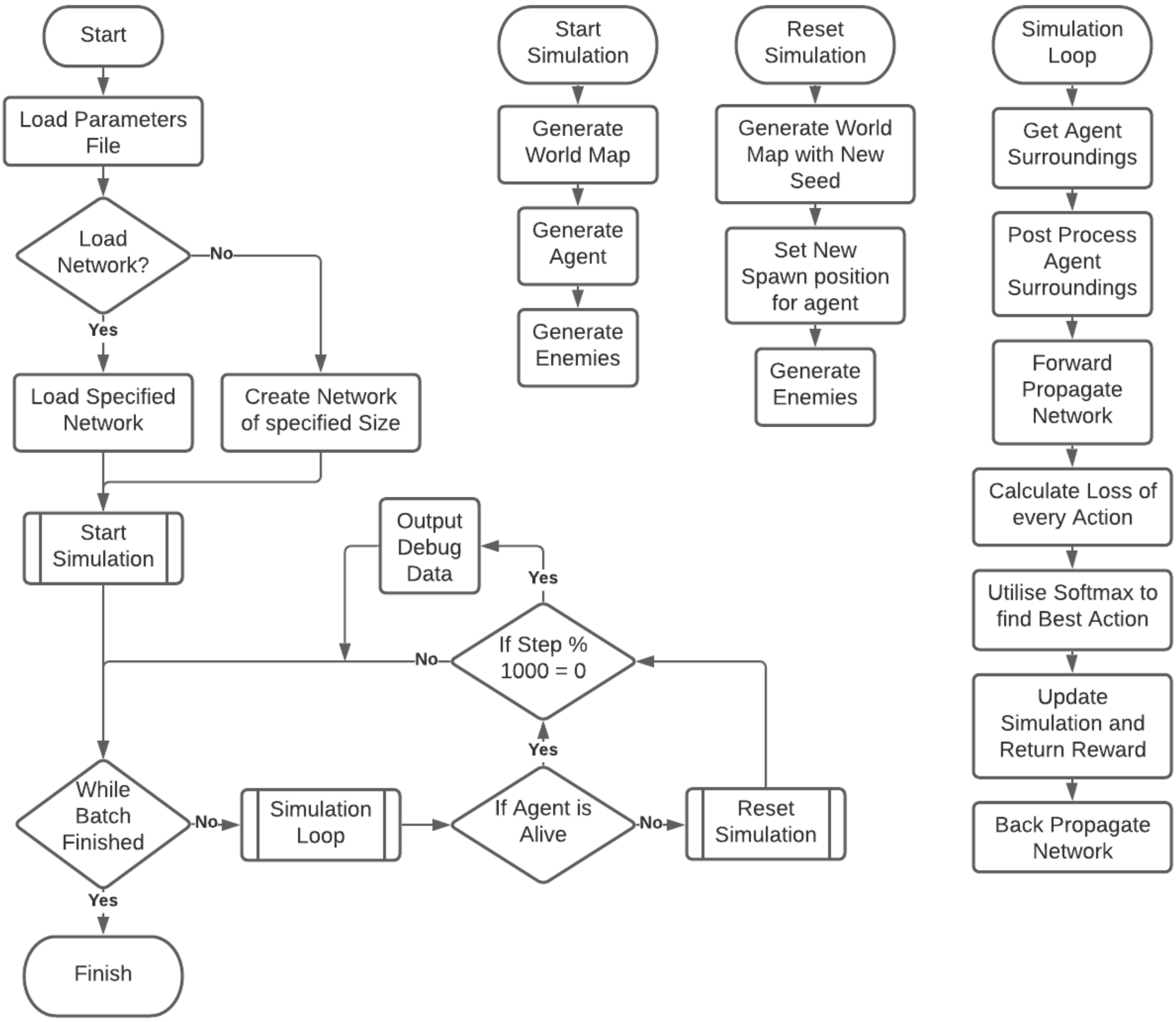
        i. ReLu
        ii. Sigmoid
        iii. SoftMax
    (d) Create methods to Forward Propagate the neural network
    (e) Create methods to calculate the loss of the network using the Bellman Equation
    (f) Create methods to Back Propagate calculated error through the neural network
    (g) Create methods to update weights and biases within the network to converge on a well trained network
    (h) Utilise the outlined Matrix class to perform the mathematical operations in the specified methods
    (i) Implement Load and Save Methods to save progress in training
    (j) Implement a Double Ended Queue/Deque Data Type
    (k) Implement Experience Replay utilising the Deque Data Type to increase training accuracy

6. Data Logger
    (a) Be able to create a Data Logger class to log data points across training
    (b) Be able to create a Data Structure for the Data Logger
    (c) Allow multiple types specified types for a single parameter
    (d) When adding a new Data Point the Logger will check it to make sure it matches the given Data Structure
    (e) Implement a Heap Data Type
    (f) Implement a Heap sort using the Heap Data Type
    (g) Be able to sort by a parameter in the Data Structure
    (h) Be able to select a single parameter from the data points
    (i) Implement Load and Save Functions to save progress during training

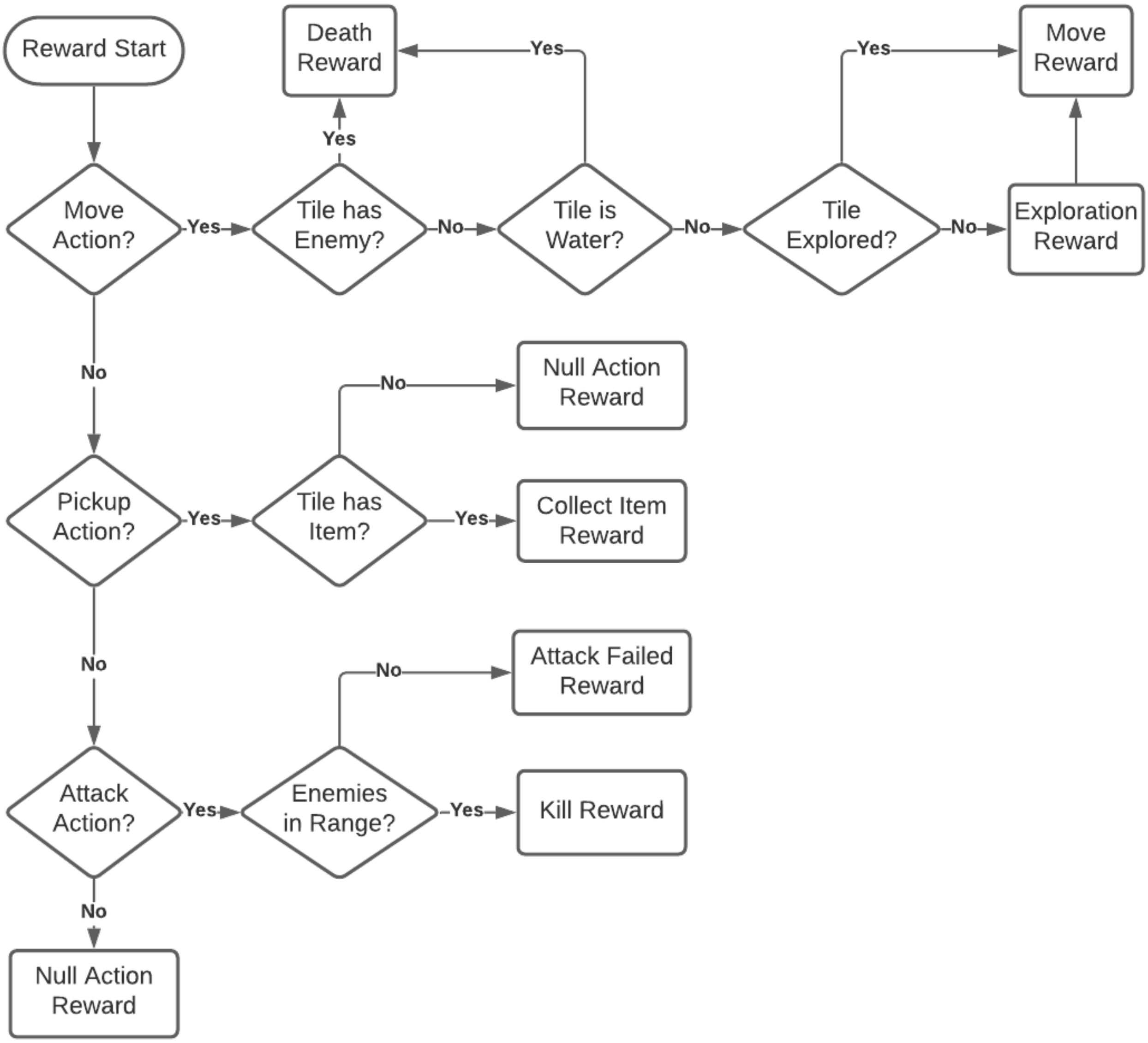
2 Design

1. System Flow Charts

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

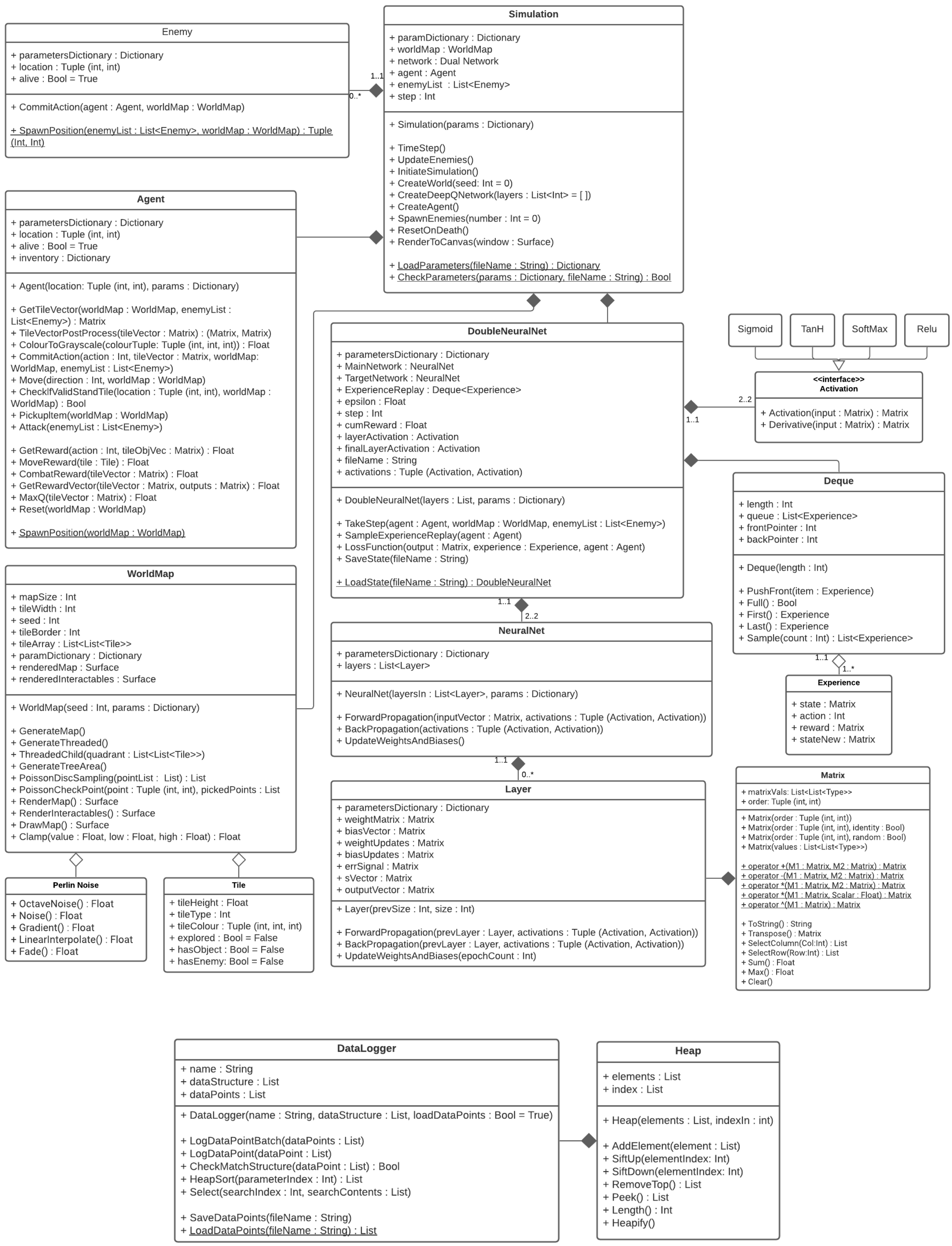


Below is shown the Action and Reward Tree for the Agent. Any Reward is added to a Total Reward Buffer and returned as part of the Function.



2. Class Diagrams Below is shown the Class Diagram of the entire Technical Solution.

   The Data Logger is listed seperately for clarity, as in practice multiple sections of the Program will aggregate with it.



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

4. 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.

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) 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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

10) Neural Network Loss Function
The algorithm for to calculate the Loss of a Dual Neural Network can calculated by 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.

11) 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.

12) 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
```

13) 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
```

14) 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
```

15) 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
```

16) 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
```

17) 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
```

18) 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
```

19) 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
```

20) 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
```

21) 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 $\lfloor \mathbf{(N\text{-}1)/2} \rfloor$ 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 **2i + 1** and **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
```

22) 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
```

23) 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
```

5. Description of Data Structures

(a) 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.

(b) 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.

(c) 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.

(d) 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.

(e) 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 **2i + 1** and **2i + 2**. The Parent of an Index is located at $\lfloor \mathbf{(i-1)/2} \rfloor$.

6. File Structure

(a) 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	0 - 1	The internally used Inventory name for collected Trees.
StartEnemyCount	Int	0 - ∞	The maximum count of Enemies to Spawn upon the creation of a new Map.
ColourWater	[Int, Int, Int]	0 - 255	The display Colour of Water.
ColourCoast	[Int, Int, Int]	0 - 255	The display Colour of Coast.

ColourGrass	[Int, Int, Int]	0 - 255	The display Colour of Grass.
ColourMountain	[Int, Int, Int]	0 - 255	The display Colour of Mountains.
ColourTree	[Int, Int, Int]	0 - 255	The display Colour of Trees.
ColourPlayer	[Int, Int, Int]	0 - 255	The display Colour of the Agent.
ColourEnemy	[Int, Int, Int]	0 - 255	The display Colour of Enemies.
MoveReward	Float	-1 - 1	The Reward Gained when the Agent Moves.
CollectItemReward	Float	-1 - 1	The Reward Gained when the Agent collects an Item.
DeathReward	Float	-1 - 1	The Reward Gained when the Agent Dies through any means.
ExploreReward	Float	-1 - 1	The Reward Gained when the Agent moves into a Tile which hasnt been Visited yet.
AttackReward	Float	-1 - 1	The Reward Gained when the Agent successfully Attacks an Enemy.
AttackFailedReward	Float	-1 - 1	The Reward Gained when the Null Action is chosen.
NoopReward	Float	-1 - 1	The Reward Gained when the Null Action is chosen.
TargetReplaceRate	Int	5 - 300	Replace Rate for Target Neural Network.
EREnabled	Int	0 - 1	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

(b) .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.

(c) .data Files

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

3 Testing

4. Testing

4.1 Testing Table

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/

1. User Input and Program Output

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

2. Matrix Implementation

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	-	-
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	-	-

3	Create Vector with List	A 1d List of any Values	Vector is created with the same values as the List	-	-
4	Print Matrix to Console	A valid Matrix of any size	Matrix Prints to the console with the correct formatting	-	-
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	-	-
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	-	-
7	Matrix Addition Calculation	Two Matrices of the same order	Matrix Addition is performed to create a new Matrix with the added values	-	-
8	Matrix Subtraction Calculation	Two Matrices of the same order	Matrix Subtraction is performed to create a new Matrix with the subtracted values	-	-
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	-	-
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	-	-
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	-	-
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	-	-
13	Matrix Transpose Calculation	A Matrix with values stored in it	New Matrix is created with values flipped across the diagonal	-	-
14	Matrix Select Column	A Matrix with values stored in it	Selects the indexed Column from the Matrix, returning as a list	-	-
15	Matrix Select Row	A Matrix with values stored in it	Selects the indexed Row from the Matrix, returning as a list	-	-
16	Vector Max in Vector	A Vector	Returns Largest value in Vector	-	-
17	Matrix Clear	A Matrix with values stored in it	Clears Matrix of any values	-	-
18	Combine Vectors	List of Vectors of the same Order	Combines the list of Vectors into a Matrix	-	-
19	Matrix Sum	-	Sums all values in the Matrix returning a $float/int$	-	-
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. Deep Q Learning Algorithm

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	-	-
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"	-	-
3	Forward Propagation Test	Where L is the Current Layer, Forward Propagation requires: $OutputVector^{L-1}$, $WeightMatrix^{L-1}$, $BiasVector^{L}$	The output of the Layer	-	-
4	Forward Propagation Multi Layer Test	Same as Entry Above	-	-	-
5	Loss Function Bellman Equation	-	-	-	-
6	Back Propagation Test	-	-	-	-
7	Back Propagation Multi Layer Test	-	-	-	-
8	Deque Push Front	A value to push to the Deque	Item is pushed to front of Deque	-	3.8
9	Deque First/Last	Call the .First() or .Last() Method for a Deque Object	Returns item at Front/Last index of Deque	-	-
10	Deque Sample N Ammount of Items	Call the .Sample(int N) Method, with a parameter of N items, for a Deque Object	Returns N number of random samples from Deque	-	-
11	Experience Replay Sampling	-	Back Propagation is performed on the sampled Deque Items	-	-
12	Activation Outputs Unit Test	Input Value Vector to the Activation Function	Returns a Vector of values, where the Activation has been applied to them	-	-
13	Activation Derivatives Output Unit Test	Input Value Vector to the Activation Derivative Function	Returns a Vector of values, where the Activation Deivative has been applied to them	-	-

4. Data Logger

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

5. Simulation

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	-	-
2	Creation of Enemies	Run program as normal with the "StartEnemyCount" Parameter >= 1	Up to the ammount of specified Enemies are created	-	-
3	Enemies Pathfind towards Agent	Run program as normal with "StartEnemyCount" Parameter >= 1	The spawned enemies pathfind towards the agnet 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	-	-
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	-	-
6	Reward System Test Basic Reward	-	Expected reward is given to agent	-	-
7	Reward System Test Complex Reward	-	Expected reward is given to agent	-	-

8	World Generates to an Acceptable Standard	Run program as normal	Generates 2d Terrain which roughly looks realistic	-	-
9	World Generation Conforms to Parameters	Utilise inputted parameters to identify the effect they have on the world Generation	Terrain changes depending on inputting Parameters	-	-
10	Perlin Noise retains Continuity	Generate two worlds with the same seed	Perlin Noise returns same value when using the same seed twice	-	-

4.2 Testing 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

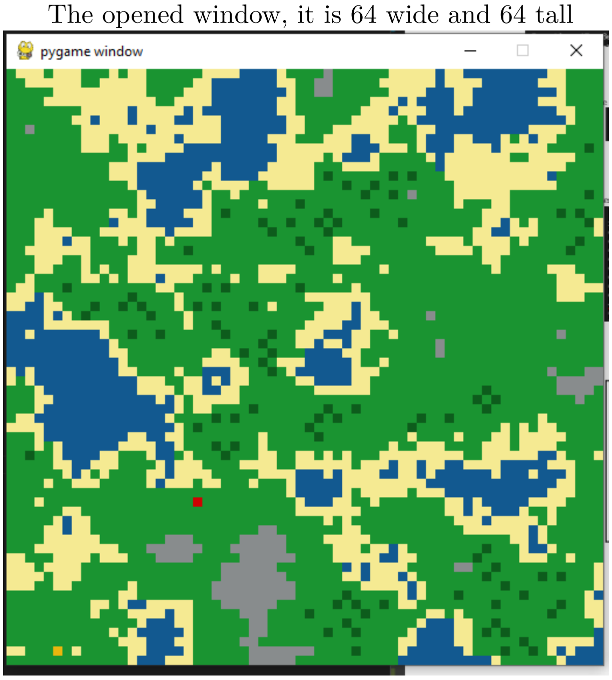
```
Load weights (Y/N): Y
State file name: DQNetwork
Created New World, Seed: 765802
Created New World, Seed: 274263
Created New World, Seed: 142187
Created New World, Seed: 613313
Created New World, Seed: 961492
Created New World, Seed: 493768
Created New World, Seed: 551641
Created New World, Seed: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

Evidence 1.6

The width/height of the window

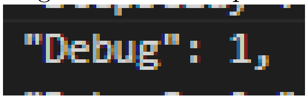
```
"WorldSize": 64,
```

The opened window, it is 64 wide and 64 tall

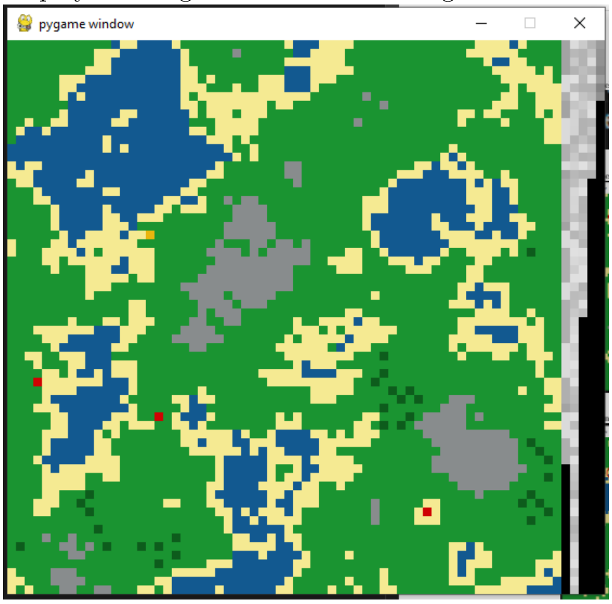


Evidence 1.7

Debug being set to 1 in the parameters file

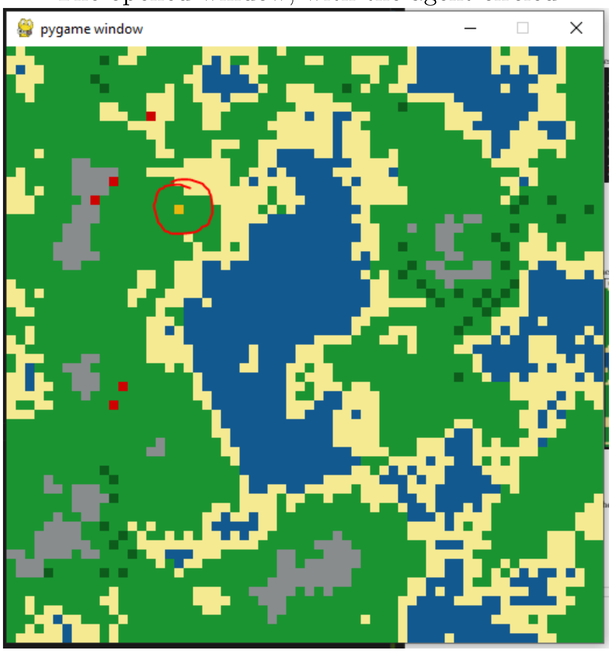


The displayed debug information to the right of the Window



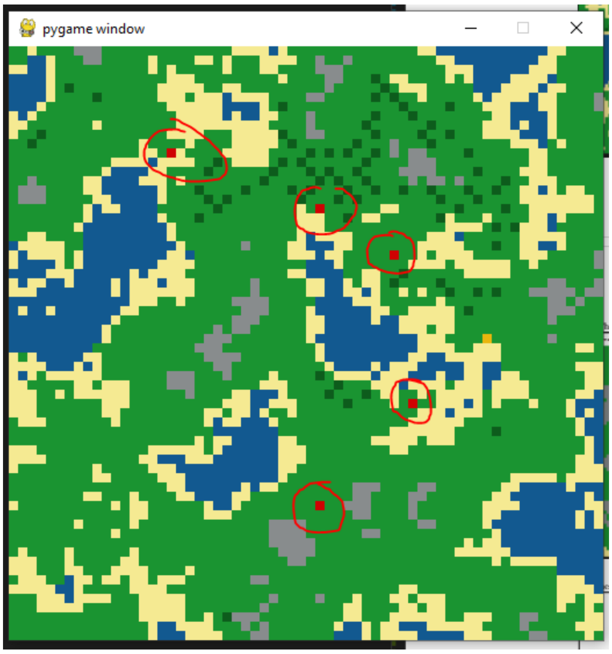
Evidence 1.8

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 2.0899999999999997 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 3.5399999999999999 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 3.3999999999999986 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
```

Evidence 2.1

Console Output, all Tests have passed with no failures

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

Evidence 2.2

Console Output, all Tests have passed with no failures

```
10000/10000 | NoMatchingInitCase
10000/10000 | UnableToCreateIdentityMat
```

Evidence 2.3

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 2.4

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
```

Evidence 3.1

Evidence 3.2

Evidence 3.3

Evidence 3.4

Evidence 3.5

Evidence 3.6

Evidence 3.7

Evidence 3.8

Pushing items to the front of the Double Ended Queue

```
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 3.9

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

```
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 3.10

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 4.1

Evidence 5.1

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

```python
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 5.2

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 5.3

Test Data Point matches struture

```
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 5.4

Test Data Point matches structure

```
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 5.5

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 5.6

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 5.7

Select Prime numbers in 1st index

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

```python
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 5.8

Test for saving a file

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

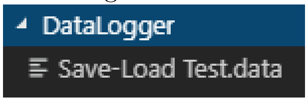
```
DataLogger
    Save-Load Test.data
```

Evidence 5.9

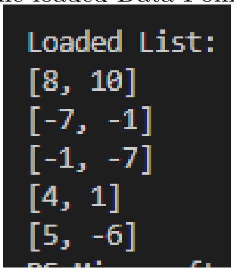
Test for loading a file

```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"



The loaded Data Points



4 Evaluation

5. Evaluation

1. Evaluation of Objectives

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

(a) Reading user inputted data

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

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

(b) Generating the Environment

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

(c) Displaying the world to a Pygame Window

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

(d) Simple Agent with a set of Actions

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

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

(e) Matrix class with Standard Operations

A Matrix can be created using 3 different methods. First using a Tuple of Integers, a new Matrix will be created of that size, with initialised 0 values. Second using a prexisting 2d list of values, a new Matrix will be created with these dimensions and values. Thirdly a 1d list of values can be used to create a 1 wide Vector of values, where it reads each value into the 1st position of each row.

All standard operations for the Matrix Object are implemented using Operator Overloading to make code less bloated. All are written efficiently utilising minimum complexity algorithms. Addition can be carried out utilising the $+$ Operator. Subtraction can be carried out utilising the $-$ Operator. Multiplication and Scalar

Multiplication are both carried out utilising the ∗ Operator. Power Operation is carried out utilising the ˆ operator. A Matrix can be converted to a Formatted String implicitly by using it in a string context.

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

(f) Creation of a Reinforcement Learning Model

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

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

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

(g) Creation of a Data Logger

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

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

2. Answering my Investigations Question

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

Can you train a Machine Learning algorithm to survive in a pseudo random, 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. This simulation

3. 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.

(a) 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 **NOT FINISHED**”

(b) Does my Tehnical 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 even 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.”

(c) 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. Evaluation of Expert Feedback

5. 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.

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

(a) 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.

(b) 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.

(c) 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

7. Technical Solution