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

1. Contents

2. Analysis

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.

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.

3. Expert

   For my expert I approached one of my friends, Ben, 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.

4. Initial Research

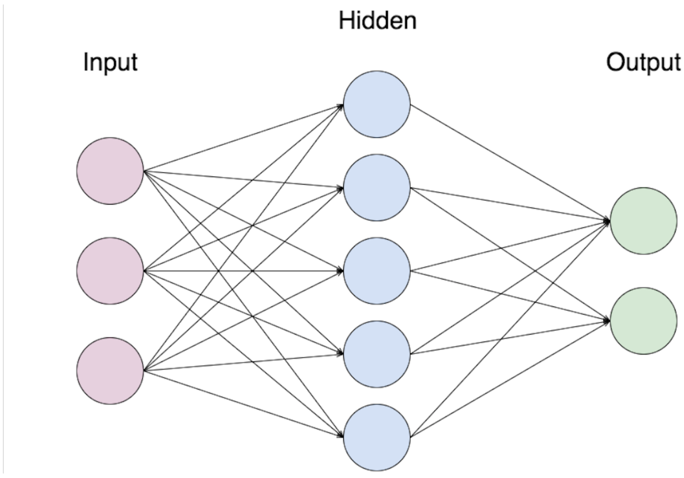
   (a) Existing Investigations

   (b) 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 $[3, 5, 2]$.

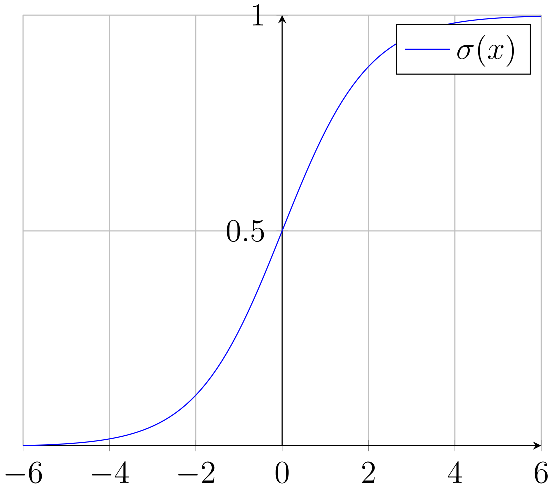


Each connection, otherwise known as an Arc or Edge, has an associated weight. Along with every output of a layer having an associated Bias. These are used to compute the outcome of a network. Forward Propagation is used to compute the outcome of a network, it has a general form and uses Matrix Multiplication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

<table>
<tr><th>Proposed Solution</th><th colspan="2">Benefits and Downsides of Proposed Solution</th></tr>
<tr><td rowspan="2">C# Graphical Libraries</td><td>Windows Forms</td><td>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.</td></tr>
<tr><td>WPF</td><td>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.</td></tr>
<tr><td>Rust</td><td colspan="2">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.</td></tr>
<tr><td rowspan="2">Rust Graphical Libraries</td><td>Piston2d</td><td>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.</td></tr>
<tr><td>Pixels</td><td>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.</td></tr>
</table>

(c) Interview

5. Prototype

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:

(a) Terrain Generation

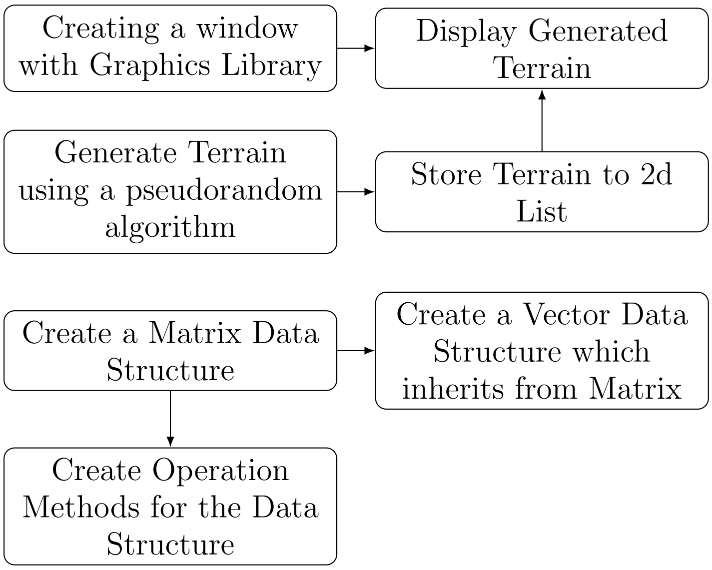
(b) Displaying the Generated Terrain using a Graphics Library

(c) 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.

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:

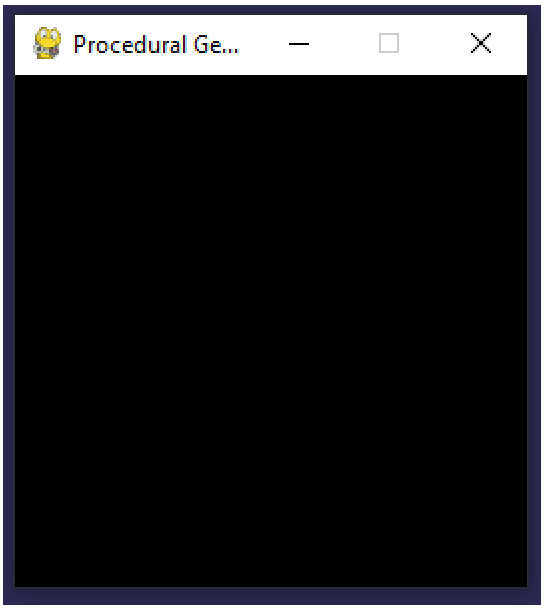
```
import pygame

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

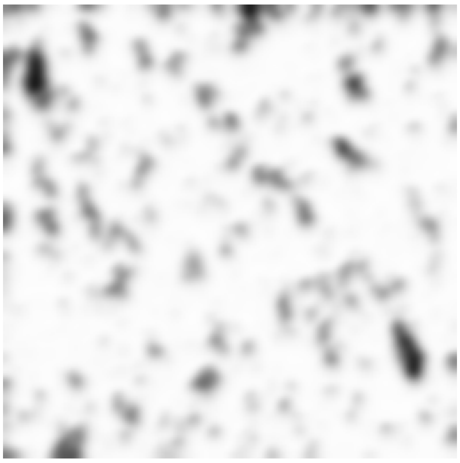
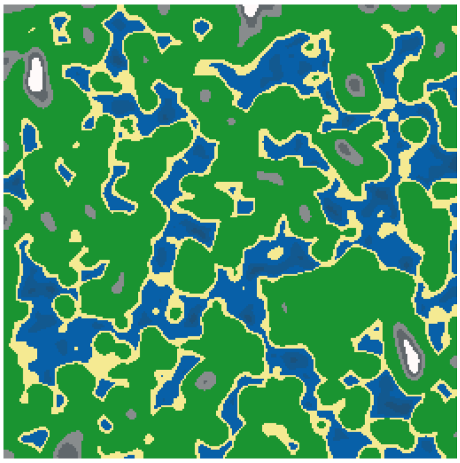
Summing these and dividing by the total grants us the average:

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

The code for this shown under the Prototype Code Header.

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.

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:

(a) 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.

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

(b) 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.

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

(c) 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.

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

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

(d) 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.

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.

6. Objectives

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

(a) User Input
   i. Read Parameters from a Json formatted file
   ii. Check Parameters fall within a certain range to prevent errors
   iii. Give user option to load Neural Network Training progress

(b) Simulation
   i. Utilise Perlin Noise to generate a 2d List of terrain heights
   ii. Store Terrain Heights in a Tile Data Type
   iii. Utilise Threading to generate Terrain Faster
   iv. Display terrain to a pygame window
   v. Map ranges of terrain heights to specific colour bands
   vi. Utilise Poisson Disc Sampling to generate objects for the Agent to interact with
   vii. Implement enemies which use basic pathfinding to traverse towards the player
   viii. Generate multiple enemies upon starting the simulation
   ix. Allow the enemies to attack the Agent

(c) Agent
   i. Implement Movement options for the Agent
   ii. Implement the ability to pick up the generated Objects
   iii. Implement the ability to attack the generated enemies
   iv. Create methods to sample the terrain around the Agent
   v. Create methods to convert the sampled Tiles into a grayscale input vector for a neural network
   vi. Create reward methods to reward the agent given the terrain samples and action

(d) Matrix Class
   i. Implement a Dynamic Matrix Class with appropriate Operations such as:
      A. Multiplication
      B. Addition
      C. Subtraction
      D. Transpose
      E. Sum
      F. Select Row/Column
   ii. Create appropriate errors to throw when utilising methods the incorrect way

(e) Deep Q Learning
   i. Dynamically create a Dual Neural Network model based upon loaded parameters
   ii. Implement an Abstract Class for Activation Functions
   iii. Implement Activation Functions inheriting from the Abstract Class such as:
      A. ReLu
      B. Sigmoid
      C. SoftMax
   iv. Create methods to Forward Propagate the neural network
   v. Create methods to calculate the loss of the network using the Bellman Equation

vi. Create methods to Back Propagate calculated error through the neural network
vii. Create methods to update weights and biases within the network to converge on a well trained network
viii. Utilise the outlined Matrix class to perform the mathematical operations in the specified methods
ix. Implement Load and Save Methods to save progress in training
x. Implement a Double Ended Queue/Deque Data Type
xi. Implement Experience Replay utilising the Deque Data Type to increase training accuracy

(f) Data Logger

i. Be able to create a Data Logger class to log data points across training
ii. Be able to create a Data Structure for the Data Logger
iii. Allow multiple types specified types for a single parameter
iv. When adding a new Data Point the Logger will check it to make sure it matches the given Data Structure
v. Implement a Heap Data Type
vi. Implement a Heap sort using the Heap Data Type
vii. Be able to sort by a parameter in the Data Structure
viii. Be able to select a single parameter from the data points
ix. Implement Load and Save Functions to save progress during training

3. Design

4. Testing

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 for testing evidence

https://thisisalink.com/youtotallybelieveme/

1. User Input and Program Output

Test No.	Test Name	Input Data / Description	Expected Output	Pass / Fail	Time Stamp
1	Loading Parameters File	Input "Default.json" file which contains the loadable values	Loads parameters into the Parameters Dictionary variable	-	-
2	Parameters within range	Input "Default.json" file	Prints to console "Parameters withing Specified Ranges"	-	-
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	-	-
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	-	-
5	Network Save 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	-	-
6	Window Opening	Run Program, enter setup info as normal	Window opens and is of the correct size/resolution	-	-
7	Window Displays correct Graphical information	Run Program, enter setup info as normal	-	-	-
8	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	-	-
9	Agent is displayed	Run Program, enter setup info as normal	Orange square displayed on screen	-	-
10	Enemies are displayed	Run Program, enter setup info as normal, with "StartEnemyCount" >= 1	Red Square/s are displayed on Screen	-	-
11	Console Messages Output	Run Program, enter setup info as normal	Console Messages Outputted per 100 Steps	-	-

2. Matrix Implementation

Test No.	Test Name	Input Data / Description	Expected Output	Pass / Fail	Time Stamp
12	Create Matrix with Tuple	-	Matrix is created with an order the same as the Tuple	-	-
13	Create Matrix with 2d List	-	Matrix is created with the same values as the 2d List	-	-

14	Create Vector with List	-	Vector is created with the same values as the List	-	-
15	Print Matrix to Console	-	Matrix Prints to the console with the correct formatting	-	-
16	Create Randomised Matrix	-	Matrix is created with randomised values between -0.5 and 0.5	-	-
17	Create Identity Matrix	-	Matrix is created with all 0's and 1's down the diagonal	-	-
18	Matrix Addition Calculation	-	Matrix Addition is performed to create a new Matrix with the added values	-	-
19	Matrix Subtraction Calculation	-	Matrix Subtraction is performed to create a new Matrix with the subtracted values	-	-
20	Matrix Multiplication Calculation	-	Matrix Multiplication is performed to create a new Matrix with the multiplied values	-	-
21	Matrix Scalar Multiplication Calculation	-	Matrix Scalar Multiplication is performed to create a new Matrix with the multiplied values	-	-
22	Vector Hadamard Product Calculation	-	Vector Hadamard Product is performed to create a new Vector with the multiplied values	-	-
23	Matrix Power Calulation	-	Matrix to the Power of is performed to create a new Matrix with the correct values	-	-
24	Matrix Transpose Calculation	-	New Matrix is created with values flipped across the diagonal	-	-
25	Matrix Select Column	-	Selects the indexed Column from the Matrix, returning as a list	-	-
26	Matrix Select Row	-	Selects the indexed Row from the Matrix, returning as a list	-	-
27	Vector Max in Vector	-	Returns Largest value in Vector	-	-
28	Matrix Clear	-	Clears Matrix of any values	-	-
29	Combine Vectors	List of Vectors of the same Order	Combines the list of Vectors into a Matrix	-	-
30	Matrix Sum	-	Sums all values in the Matrix returning a $float/int$	-	-
31	Randomised Matrix Constructor Tests	Generator Constructor Parameters randomnly for 10000 Tests	All Tests Should produce a valid Matrix	Pass	-
32	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	-
33	Randomised Operator Tests	Generator Random Data to test the Operator Methods for 10000 Tests	All Tests should produce the correct result	Pass	-
34	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	-

3. Deep Q Learning Algorithm

Test No.	Test Name	Input Data / Description	Expected Output	Pass / Fail	Time Stamp
35	Networks are Created	Run Program, enter setup info, denying the loading of weights	A Dual Neural Network is created after Program Start	-	-
36	Networks conforms to Parameters	Run Program, enter setup info, denying the loading of weights	The created Dual Network conforms to the specified structure in the parameter "DeepQLearningLayers"	-	-
37	Forward Propagation Test	-	-	-	-
38	Forward Propagation Multi Layer Test	-	-	-	-
39	Loss Function Bellman Equation	-	-	-	-
40	Back Propagation Unit Test	-	-	-	-
41	Back Propagation Multi Layer Unit Test	-	-	-	-
42	Deque Push Front	-	Item is pushed to front of Deque	-	-
43	Deque Pop	-	Front item is removed and returns from Deque	-	-
44	Deque First/Last	-	Returns item at Front/Last index of Deque	-	-
45	Deque Sample N Ammount of Items	-	Returns N number of random samples from Deque	-	-
46	Experience Replay Sampling	-	Back Propagation is performed on the sampled Deque Items	-	-
47	Activation Outputs Unit Test	-	-	-	-
48	Activation Derivatives Output Unit Test	-	-	-	-

4. Data Logger

Test No.	Test Name	Input Data / Description	Expected Output	Pass / Fail	Time Stamp
49	Heapify Method	-	Returns Binary Tree to have the Heap Property (Such that for any node the child nodes are <= to it)	-	-
50	Heap Sort Decending	-	Sorts the list of items into Descending order	-	-
51	Add Point	-	Point is added to Data Points list	-	-
52	Match Data Struture with Single	-	No error thrown	-	-
53	Match Data Struture with Multi-Typed	-	No error thrown	-	-

54	Match Data Struture with List-Typed	-	No error thrown	-	-
55	Match Data Structure Error	-	Error is thrown with correct info	-	-
56	Select Query	-	Returns a list of the selected column where the Search Contents Matches	-	-
57	Load Data Points	-	Loads Data Points from specified File	-	-
58	Save Data Points	-	Saves Data to specified File	-	-

5. Simulation

Test No.	Test Name	Input Data / Description	Expected Output	Pass / Fail	Time Stamp
59	Creation of Agent	-	Agent is created as an instance of the Agent Class	-	-
60	Creation of Enemies	-	Up to the ammount of specified Enemies are created	-	-
61	Enemies Pathfind towards Agent	-	The spawned enemies pathfind towards the agnet using the defined pathfinding algorithm	-	-
62	Getting Tile Data	-	Returns a Vector of the surrounding tile objects	-	-
63	Convert Tile Data	-	Converts Tile Data into two vectors, Grayscale Colour and Tile Type	-	-
64	Reward System Test Basic Reward	-	Expected reward is given to agent	-	-
65	Reward System Test Complex Reward	-	Expected reward is given to agent	-	-
66	World Generates to an Acceptable Standard	-	Generates 2d Terrain which roughly looks realistic	-	-
67	World Generation Conforms to Parameters	-	Terrain changes depending on inputting Parameters	-	-
68	Perlin Noise retains Continuity	-	Perlin Noise returns same value when using the same seed twice	-	-

5. Evaluation

6. Prototype Code

Below is the code I created while developing my Prototype.
The 3 Scripts listed in order are:

1. main.py
2. worldClass.py
3. mathLib.py

---

1. main.py

```
#Imports
import pygame, random, json, os, time
from datetime import datetime
import worldClass, agentClass, mathLib

#Variables
simSize = 64
gridSize = 4
simSeed = 420

#World Functions
def DrawWorld():
    if world.grayscale == False:
    for y in range(0, simSize):
        for x in range(0, simSize):
        colour = world.colourArray[x][y]
        pygame.draw.rect(window, (colour), ((x * gridSize), (y * gridSize), gridSize,
    else:
    for y in range(0, simSize):
        for x in range(0, simSize):
        value = world.heightArray[x][y]
        pygame.draw.rect(window, (255 * value, 255 * value, 255 * value), ((x * gridSi


#World Gen Functions
def RandomWorld():
    SetWorld(random.randint(0, 10000))
def SetWorld(seed):
    world.GenMap(seed)
    DrawWorld()

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

world = worldClass.WorldMap(simSize)
RandomWorld()

#Main loop
running = True
while running == True:
    for event in pygame.event.get():
    if event.type == pygame.QUIT:
        running = False
    elif event.type == pygame.KEYDOWN:
        if event.key == pygame.K_RETURN:
        RandomWorld()
        elif event.key == pygame.K_F2:
```

```
        pygame.image.save(window,"DevelopmentScreenshots\\screenshot{}.png".format(len

    pygame.display.update()
```

2. worldClass.py

```
import random, json

class WorldMap():
    def __init__(self, size):
        self.arraySize = size
        self.heightArray = [[-1 for i in range(size)] for j in range(size)]
        self.colourArray = [[(0, 0, 0) for i in range(size)] for j in range(size)]
        self.typeArray = [[-1 for i in range(size)] for j in range(size)]

        self.inverted = False
        self.grayscale = False
        self.upNeutralDown = 0
        self.averaging = 0
        self.params = []
        self.thresholds = []
        self.LoadParameters("DefaultParameters.json")

    def LoadParameters(self, fname):
        file = open("Presets\\{}".format(fname), "r")
        self.params = json.loads(file.read())
        file.close()

        for key in self.params:
            if key == "Inverted":
                if self.params[key] == 1:
                    self.inverted = True
            elif key == "UpNeutralDown":
                self.upNeutralDown = self.params[key]
            elif key == "Averaging":
                self.averaging = self.params[key]
            elif key == "Grayscale":
                if self.params[key] == 1:
                    self.grayscale = True
            else:
                self.thresholds.append((float(key),(self.params[key][0], self.params[k

    def ConvertTypes(self):
        for y in range(0, self.arraySize):
            for x in range(0, self.arraySize):
                for i in range(len(self.thresholds)):
                    value = self.heightArray[x][y]
                    if self.inverted:
                        value = 1 - value
                    if value <= self.thresholds[i][0]:
                        #print(thresholds[i][0])
                        self.colourArray[x][y] = self.thresholds[i][1]
                        self.typeArray[x][y] = i
                        break


    def GenMap(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)
```

```
        for i in range(self.upNeutralDown):
            self.UpNeutralDownGen()
            #print("UNDGen")
        for i in range(self.averaging):
            self.AverageGen()
            #print("averaging")

        self.ConvertTypes()

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

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

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

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

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

        self.heightArray = dupMap

    def AverageGen(self):
        dupMap = self.heightArray
        for y in range(0, self.arraySize):
            for x in range(0, self.arraySize):
                total = 0
```

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

                dupMap[x][y] = total / count
        self.heightArray = dupMap

    def Clamp(self, val, low, high):
        return low if val < low else high if val > high else val
```

3. mathLib.py

```
import math, random
class Matrix():
    def __init__ (self, Values, cols = 0, identity = False):
        if type(Values) == list: # Predefined Values
            self.matrixArr = Values

        elif identity == True: # Identity Matrix
            if Values != cols:
                raise Exception("Cant create Identity Matrix of different orders")
            else:
                self.matrixArr = [[0 for i in range(cols)] for j in range(Values)]
                for y in range(0, Values):
                    self.matrixArr[y][y] = 1

        elif Values > 0 and cols > 0: # Blank Matrix of size x by y
            self.matrixArr = [[0 for i in range(cols)] for j in range(Values)]

        else: # Error Creating Matrix
            raise Exception("Error Creating Matrix")

    def Val(self):
        return self.matrixArr

    def Dimensions(self):
        return [len(self.matrixArr), len(self.matrixArr[0])] # Rows - Columns

    def ScalarMultiply(self, multiplier):
```

```
        for y in range(0, len(self.matrixArr)):
            for x in range(0, len(self.matrixArr[0])):
                self.matrixArr[y][x] = self.matrixArr[y][x] * multiplier

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

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


    def SubMatrixRange(self, y1, y2, x1, x2):
        subMat = Matrix(y2 - y1 + 1, x2 - x1 + 1)
        for y in range(y1, y2 + 1):
            for x in range(x1, x2 + 1):
                subMat.matrixArr[y][x] = self.matrixArr[y][x]
        return subMat

    def RandomVal(self):
        self.matrixArr = [[random.randint(1, 100) for i in range(self.Dimensions()[1])

    def ConvertToVector(self):
        return Vector(self.matrixArr)

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

```python
        return det

def det(m):
    top_length = len(m[0])
    height = top_length - 1
    submats = []

    for i in range(0, top_length):
        submat = [[] for i in range(height)]
        for j in range(0, top_length):
            if i != j:
                for k in range(height):
                    submat[k].append(m[k+1][j])
        submats.append(submat)
    return submats

# Static Methods
@staticmethod
def MatrixAddSubtract(m1, m2, subtract = False): # Dont know how else i would make
    m1Dims = m1.Dimensions()
    m2Dims = m2.Dimensions()
    if m1Dims[0] != m2Dims[0]:
        raise Exception("Matrices Row Order does not match")
    elif m1Dims[1] != m2Dims[1]:
        raise Exception("Matrices Column Order does not match")
    elif type(m1) != type():
        raise Exception("Types do not match, Convert Vector to Matrix or vice vers
    else:
        newMat = Matrix(m1Dims[0], m1Dims[1])
        for y in range(0, m1Dims[0]):
            for x in range(0, m1Dims[1]):
                if subtract:
                    newMat.matrixArr[y][x] = m1.Val()[y][x] - m2.Val()[y][x]
                else:
                    newMat.matrixArr[y][x] = m1.Val()[y][x] + m2.Val()[y][x]
        return newMat

@staticmethod
def MatrixMultiply(m1, m2): # Not that efficient, needs optimisation
    m1Dims = m1.Dimensions()
    m2Dims = m2.Dimensions()
    if m1Dims[1] != m2Dims[0]:
        raise Exception("Matrices Multiplication Error")
    else:
        if(type(m2) == Vector):
            newMat = Matrix(m1Dims[0], m2Dims[1])
        else:
            newMat = Matrix(m1Dims[0], m2Dims[1])
        for row in range(0, m1Dims[1]):
            subRow = m1.Val()[row][0:m1Dims[1]]
            for col in range(0, m2Dims[1]):
                subCol = []
                for i in range(0, m1Dims[0]):
                    print(i)
                    subCol.append(m2.Val()[i][col])
                total = 0
                for x in range(0, len(subRow)):
                    total += subRow[x] * subCol[x]
                newMat.matrixArr[row][col] = total
        return newMat
```

```
class Vector(Matrix):
    def __init__(self, val):
        if type(val) == list:
            if len(val[0]) != 1:
                raise Exception("Invalid Vector, use Matrix Instead")
            else:
                self.matrixArr = val
        else:
            self.matrixArr = [[0 for i in range(1)] for j in range(val)]

    def ConvertToMatrix(self):
        return Matrix(self.matrixArr)

    @staticmethod
    def DotProduct(v1,v2):
        if type(v1) != Vector or type(v2) != Vector:
            raise Exception("Wront Types:{},{} passed into Dot Product".format(type(v1
        else:
            total = 0
            for i in range(v1.Dimensions()[0]):
                total += v1.Val()[i][0] * v2.Val()[i][0]
            return total
```

7. Technical Solution