An Investigation into Machine Learning and Neural Networks through the Simulation of Human Survival

Computer Science NEA

Name: Matthew Brabham
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 an Agent will be controlled by a Machine Learning algorithm. The survival simulation will present multiple challenges 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
3. Expert
4. Initial Research

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

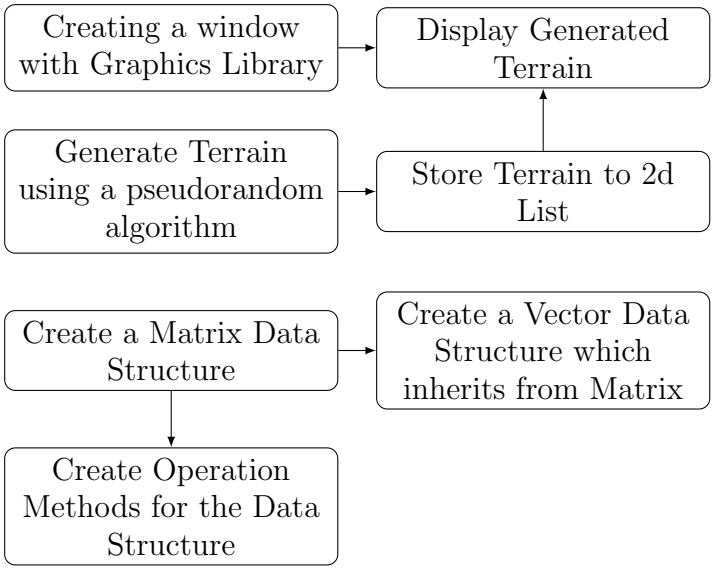
(b) Displaying the Generated Terrain using a Grahics Library

(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.

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:

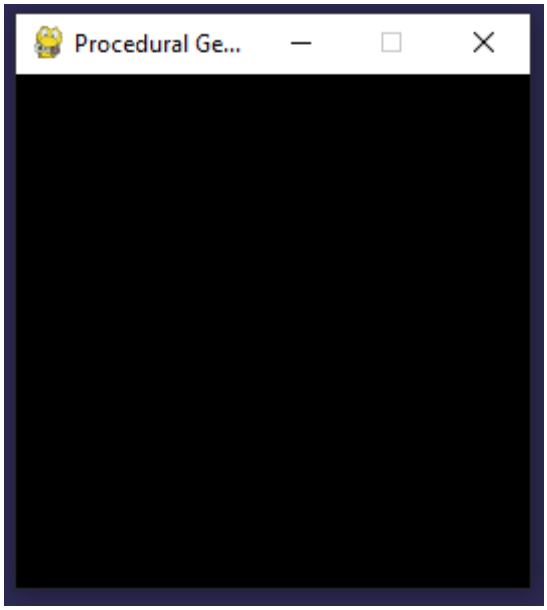
```
import pygame

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:

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

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

Average Method

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
    (a) Parameter Loading
    (b) Neural Network Loading
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

Below is the included NEA Testing video for testing evidence

https://thisisalink.com/youtotallybelieveme/

1.1 User Input

Test No.	Test Name	Input Data	Expected Output	Pass / Fail	Time Stamp
1	Loading Parameters File	"Default.json" file which contains the loadable values	Loads parameters into the Parameters Dictionary variable	Pass	00:00
2	Test Parameters within range	-	-	-	-
3	Loading of Network Weights	-	-	-	-
4	-	-	-	-	-
5	-	-	-	-	-