

Computer Games Development

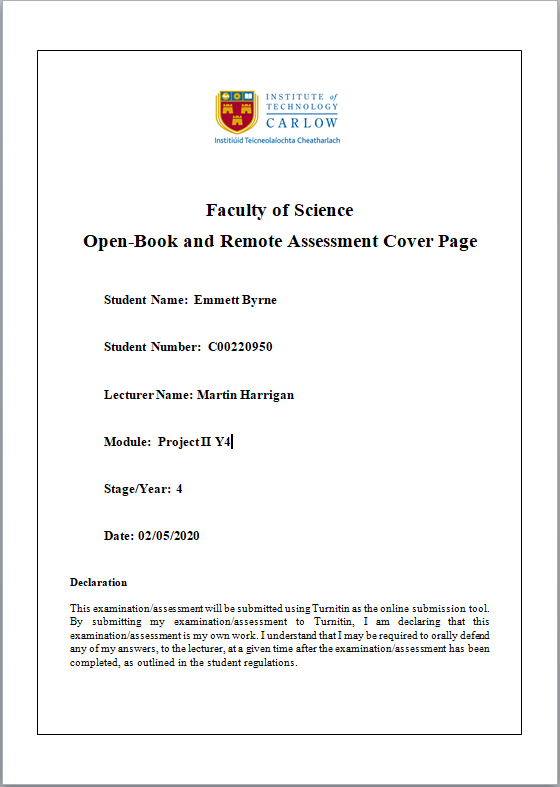
Project Report

Year IV

Emmett byrne

C00220950

Submitted: 3rd May 2020



**Contents**

[Project Abstract](#_ggvskvmn03lf) **2**

[Project Introduction and Research Question](#_if6ogk1xv1qo) **3**

[Background](#_8wqbbc2yuko) **4**

[Literature Review](#_2et92p0) **4**

[Project Description](#_4u13uhgna50m) **7**

[Study](#_9nhysnj9p4ab) **9**

[Project Milestones](#_fahibbh5tqzj) **11**

[Results and Discussion](#_4d34og8) **12**

[Success Rates:](#_mg7e2b55617j) 12

[Moves Taken:](#_jmpx30lmu0ny) 13

[Time taken to train and complete maze:](#_v3mzrad8rzgv) 14

[Project Review and Conclusions](#_58tvzekf6f3k) **16**

[References](#_17dp8vu) **16**

# Project Abstract

My project is to create a neural network and investigate its capabilities to solve a 2D maze with very limited information. The maze is built in Python and can be rendered out using PyGame and the Neural network is built with Tensorflow and uses Q learning a reinforcement learning algorithm to teach itself how to solve the maze. The maze is randomly generated and the amount of walls in the maze varies based on it’s difficulty level.

I created an experiment to test the Neural networks capabilities by getting it to train continuously on randomly generated mazes and increasing the difficulty over time until the maze is too difficult for it to complete. Once the experiment has finished we generate metrics and graphs and are able to come to a number of interested conclusions.

# Project Introduction and Research Question

For my project I want to research a neural networks ability to complete a simple 2D maze when it’s given limited information about the course.The maze I want to have the Neural networks complete is a simple procedurally generated 2D grid of different types of tiles where the network must move from tile to tile in order to reach its goal, made in Python using the TensorFlow and Pygame libraries. The objective of this is to create a data set from the results so that we can assess if we can create a neural network that can accurately complete the mazes we give it.

The main reason I wanted to work on this project was that in nearly every example I’ve seen of a Neural network trying to complete a maze it would have perfect information of the entire maze and I always found this odd since using a pathfinding such as A\* is a much better solution to these problems which is why I wanted to do something similar but give the Neural network very limited information of its surroundings since I think this kind of situation is much more suited to machine learning.

# 

# Background

Before starting this project I had practically no experience at all in working with neural networks. However, I have always had a great interest in making a neural network as I have always enjoyed reading and watching sources of neural networks solving interesting problems.

From these sourcesI got the idea of creating a neural network to solve a maze. I've seen many examples of neural networks solving mazes but I always found it strange that in each of these examples the neural network had perfect information about the maze and from this I decided it would be much more relevant to machine learning if the neural network had very limited information about the maze.

All of the technology I used in this was completely new to me as I had never used either TensorFlow or Pygame. Fortunately I did have some limited experience with Python having made some simple projects in the year prior to starting this project.

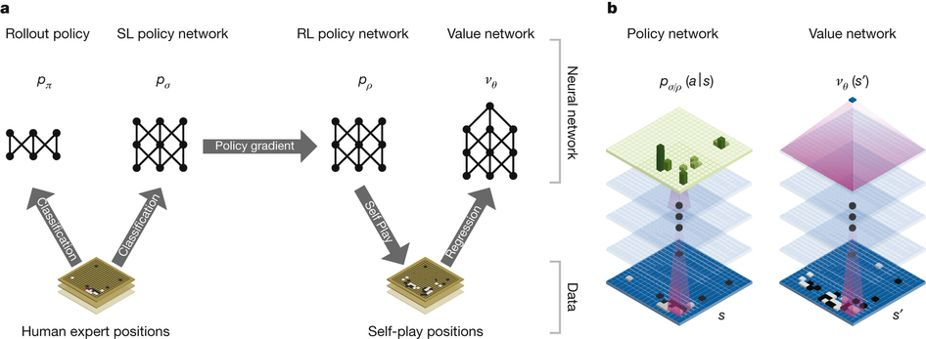
# Literature Review

*Silver David, Huang Aja, nature.com, 27 January 2016, november 2019,* [*https://www.nature.com/articles/nature16961*](https://www.nature.com/articles/nature16961)

This Article was published in the Nature magazine on the 27th of January 2016.

The article teaches a neural network to play the classic game of Go, which is normally very difficult for Neural Networks to play due to it’s enormous complexity. The complexity of the game is measured by its breadth (The number of possible legal moves per position) and its depth (The number of turns per game) which gives Go a breadth of 250 and a depth of 150. As a comparison chess would have a breadth of 35 and a depth of 80.

In order to create a Neural Network capable of playing Go the authors first train a supervised learning(SL) policy network using moves made by human experts and then trains a reinforcement learning policy to improve the SL network by getting it to play games against previous iterations of itself. A value network is also trained to predict the winner of games.



*Violante Andre, towardsdatascience, Mar 18, 2019, may 2020,* [*https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56*](https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56)

This Articles explains Q-learning and how it works, it goes through what Q-learning is, how to create a Q-Table and breaks down the Q-learning algorithm

The article states that Q-learning is a reinforcement learning algorithm that tries to find the best action given the current state and seeks to maximise the total reward.

The article goes on to explain that a Q-table is a list of [states, actions] which stores our q-values, this becomes a reference point for a Neural network to select the best action.

Next the article explains the learning process. The process starts with the agent taking an action which is either random or based upon the Q-table. Once the Neural network takes an action the results of that action is passed into the algorithm and the q-table is updated.

The following is the equation used in the article to update the Q-values in it’s Q-table:

*Q[state, action] = Q[state, action] + lr \* (reward + gamma \* np.max(Q[new\_state, :]) — Q[state, action])*

The article then explains the variables in this equation starting with lr which stands for learning rate, this is used to adjust how much the Neural network values the new Q-value over the old Q-value.

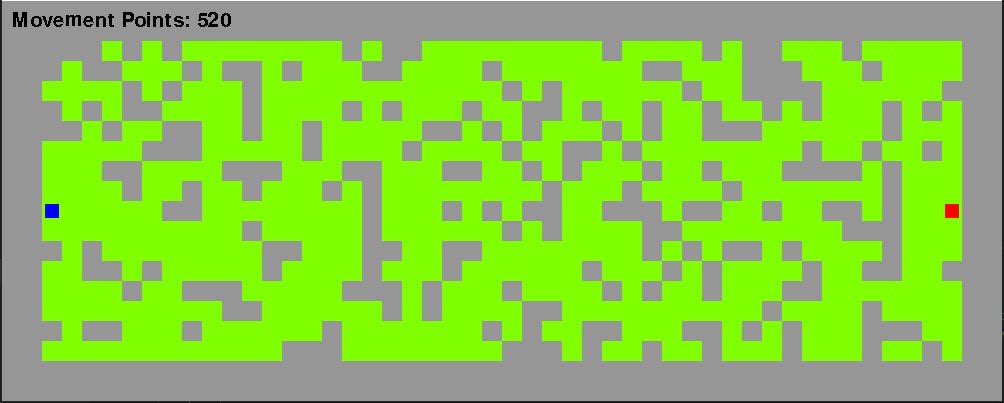
Next is gamma also known as the discount factor, the purpose for gamma is to balance the immediate and future reward.

The last variable explained is the reward. This is simply a value given to the action that was just taken.

# Project Description

The maze I will be creating will be a procedurally generated 2D grid of various different tiles. The character will start at the left of this grid and attempt to work across to the right of the grid. There will be a move limit that will make the Neural network fail the course if it takes too many moves. The maze will have multiple levels of difficulty where the higher the difficulty will result in more obstacles appearing in the maze

*Example of the maze*



For the neural network we will be using a library called Tensorflow which uses python. With this we will create a model for our network which will consist of 15 input neurons, 13 of these input neurons will represent every tile within 2 moves of the network in the maze and the last 2 neurons will represent the current position of the network. Finally we will have 4 output neurons to represent each cardinal direction and we will take the neuron with the greatest value as the network's decision. The Neural network will be trained on the maze using Q-Learning which is a reinforcement learning algorithm where the network updates it’s weights and biases based on the reward it receives from transitioning from one state to another

The last major part of my project was the experiment which was created to evaluate the success of the network. The experiment works by starting with a maze of difficulty 0 and training the network on 1000 randomly generated mazes at that difficulty where the network can learn from its actions and then once the network has had enough time to train it will be given 100 more mazes to complete but this time it will not be learning and we use these to determine how successful the Neural network is at completing the maze. Once it was completed the 100 mazes the difficulty level of the maze will be increased and the entire process will be repeated continually until the Neural can no longer successfully complete the maze. Once the Neural network has finished we can then output the results of the experiment in the form of graphs. The results for this experiment can be found in the Results and Discussion section.

When I first started on this project I had absolutely no experience with Neural networks and very little experience with python so I was very happy with what I managed to achieve. The most difficult part of my project was trying to implement the Q-Learning algorithm which took me weeks of work to actually get working correctly.

# 

# 

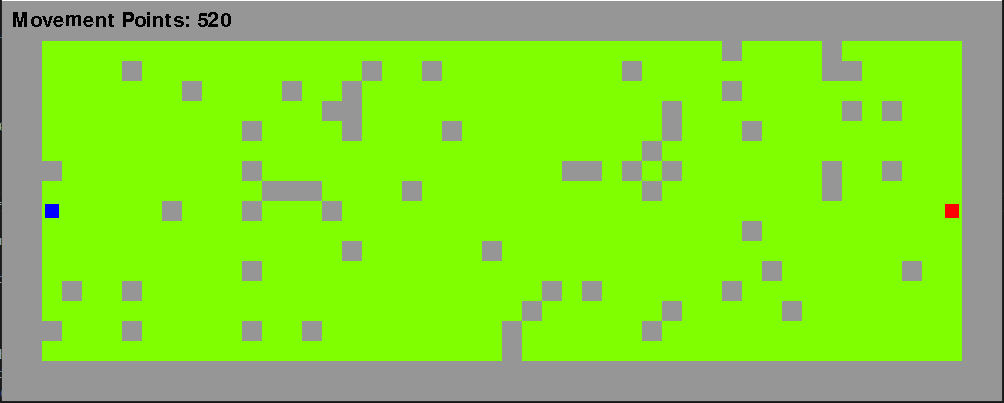
# Study

In order to evaluate how successful my project was I created an experiment. The objective of this experiment is using the neural network I have created to learn and complete a series of increasingly more difficult mazes and to find out at what point do the mazes become far too complicated for the neural network to complete.

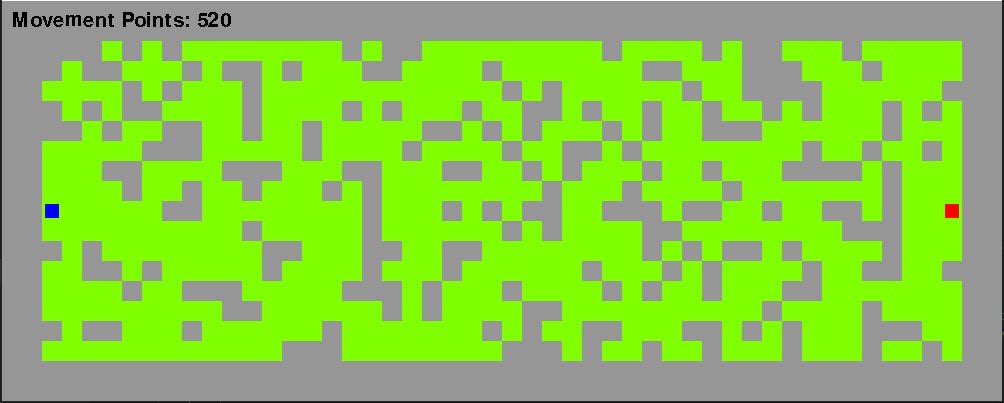
There will be two versions of this network used in this experiment. The first simple version is a network that only has fully connected input and output layers of neurons. The second network will be more complex and will have an input layer, a fully connected hidden layer and a fully connected output layer.

The difficulty level of the maze is the % chance that a wall is placed when the maze is being generated. The % chance of a wall will be the (difficulty level)\*(3), so for example at a difficulty level of 2 then every tile will have a 6% chance of being a wall

**Example of difficulty at level 2:**

****

**Example of difficulty at level 10**



The experiment will start at a difficulty level of 0 and if the neural network successfully learns to complete the maze at that difficulty then we will increase the difficulty by 1 and repeat the process until the neural network can no longer complete the maze

For each difficulty the neural network will first have to learn how to beat it at that difficulty. To do this we will simply generate a maze at that difficulty and have the neural network try to complete the maze and learn from it’s actions, we then repeat this process 1000 times generating a completely new maze each time.

Once the neural network has trained itself at a difficulty level we must now prove whether or not it is able to successfully complete the. To do this we will simply generate a maze and have the neural network try to reach the goal within the move limit and for the sake of accuracy we will do this 100 times generating a new maze each time. This will hopefully give us an accurate idea of how well the Neural network can complete the maze.

From this experiment I intend to capture a number of metrics that we will be able to use to evaluate the success of the Neural networks after they have completed the experiment. The metrics I will be taking will be:

* The average number of moves the neural network took at each difficulty.
* The % rate it succeeded at completing the maze at each difficulty.
* The amount of time it took to complete the experiment.

From my observations prior to this experiment I would like to make some general predictions as to the results of this experiment.

My first prediction is that the more complex network will reach a higher difficulty than the simple network but not by a lot. This is because the decision making for the network isn’t very complex due to its limited range of vision.

Next and this will seem quite obvious but the number of attempts needed to complete a maze will directly correlate to the difficulty of the maze as will also the % success rate.

# Project Milestones

My first major deadline was to do some initial research into my project. I wanted to find what libraries I could use, learning algorithms, etc. I started researching in early October and I gave myself a deadline of mid-October.

After I had completed my research I started working on creating a Neural Network with the OpenNN library and also creating a maze in C++ for that network to solve. I planned on working on this from late October to early December. However, I stopped working on this in late November due to the difficulty I was experiencing working with the OpenNN library.

After the difficulty I faced with OpenNN I decided to switch to TensorFlow and Python and make a very simple test network to see how feasible it was to make the neural network I wanted. I started this in late November and finished making my network early December.

Next I worked on creating a maze for the neural network to train in and rendering it out with the Pygame library. I started this before the Christmas Holidays and finished working on it halfway through the Holidays before my surgery on the 19th of December.

Once I had returned from the Christmas holidays I began working on creating a neural network that could learn how to complete my maze. I knew this would be one of the more difficult parts of my project so I gave myself until the end of February to work on this. I managed to successfully create a neural network that could learn to finish my maze but it was not as good as I wanted it to be so I took a bit more time to improve it until I felt it was competent enough at completing the maze and finished working on this at about mid-March.

The final part of the project I had to work on was to create an experiment to show my findings and provide some insight and metrics to what I found in the project. As this was the final thing I worked on this up until a week before the final deadline of the project.

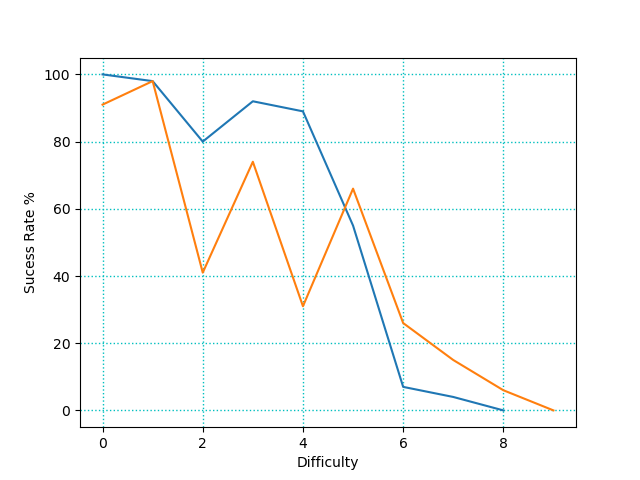
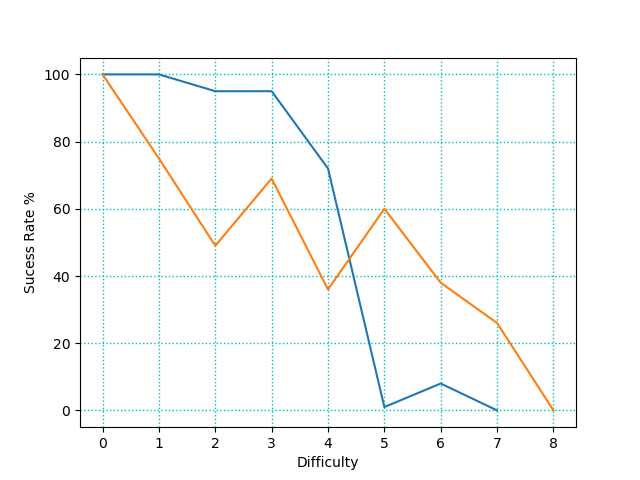
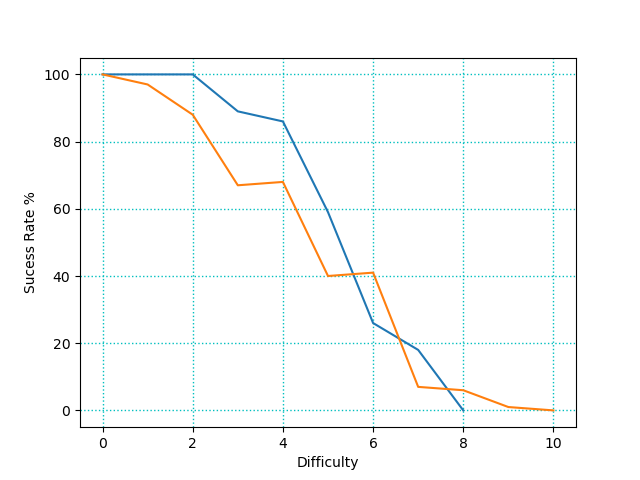
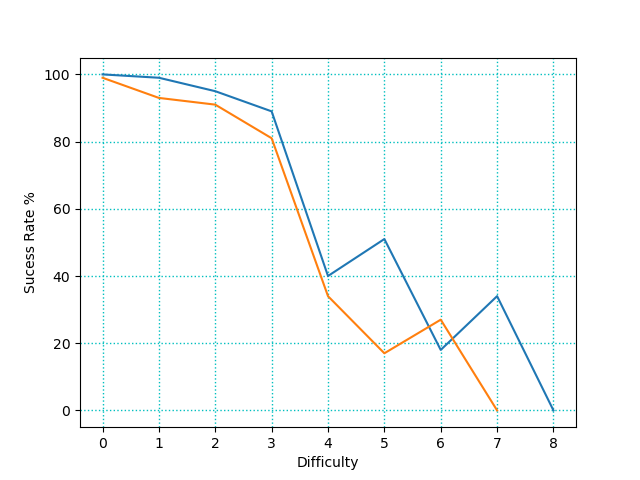
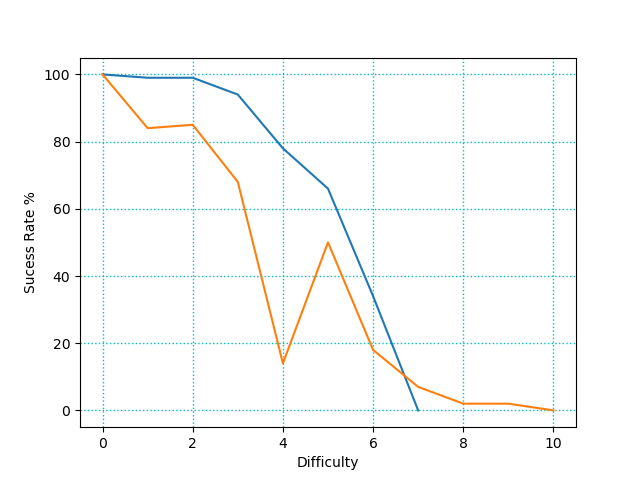
Throughout the project I felt I made good steady progress over the academic year towards completing each milestone. I did however find it very difficult to make hard deadlines for specific dates due to my complete inexperience with neural networks and how hard it would be to create and train one so I had no context for how much work it would need.  
The one major deadline I set myself when I first started working on the project was for the end of January to have a working Neural network that could learn and solve a maze however due to the difficulties I had working with OpenNN and how hard I was finding it to work with the injuries to my wrist I changed that deadline to the end of February.

# Results and Discussion

I ran the whole experiment 5 times so I could get more accurate results and generated the following metrics from the 5 experiments.

The Blue line on the graph represents the more complex network and the orange line represents the simple network

## Success Rates:



## Moves Taken:

# 

# 

## Time taken to train and complete maze:

# 

From the data we generated I’ve made a few interesting observations, while there were many results that I expected there were some very surprising results that I didn’t expect at all.

To start with the completely obvious expected results we can clearly see that as the maze gets more difficult the Neural networks start to struggle and their success rate drops and the amount of moves they take to complete the maze increases. From the success rate graphs we can clearly see that both Neural networks can get to difficulty level 5 (15% chance for each tile to be a wall) without much issue after that however their success drops quite quickly until they can no longer finish the maze.

A result that i did not expect at all was that the simple network would reach higher difficulties than the complex network. From my observations of the networks I believe this is because the more complex network becomes much more cautious and will avoid walls in favor of open spaces but as the difficulty of the maze rises and more and more walls are added this behaviour works against the complex network.

Another strange result I had was that the data I got from the simple network was very sporadic and inconsistent. I speculate that this is due to it learning a bad behaviour and then later getting out of that behaviour.

The time taken to train and complete the maze was incredibly consistent, I got almost the exact same shape and results every time I ran the experiment. What was surprising about this was that the complex network took less time on earlier difficulties and more time on the later difficulties. This I think is due to the complex being more efficient in early difficulties as we can see from the number of turns taken but as the success rate drops for both networks this makes the simple network faster and takes less time since a higher success rate means completing the maze faster and thus less time taken. Since the more complex network has the extra fully connected layer the learning process takes a bit more time as we can clearly see from the very height of the graph where the complex will always reach a much higher time as compared to the simple network.

From observing the network during and after it’s training I made a few interesting observations.

One observation I made was that both networks as they trained would start to favor either the top or bottom half of the maze and they would become much better at solving the maze from their favored half as opposed to their weak half. This reminded me of how people are either right or left handed.

I also observed that the complex network would develop some more complex behaviours, for example it would start to avoid what I call the horseshoe trap which was a group of walls in the shape of a horseshoe that if the network wandered into would struggle to get out of. I also noticed that it would learn to favor open spaces instead of tight corridors

# Project Review and Conclusions

Overall I feel that this project was a success. The main part of my project was to create a network that could solve a maze and that’s exactly what I accomplished and I created a nice experiment that generated a lot of data that led to donme interesting results and finding. A lot of these results I did not expect, especially how the more complex behaviour of one network over the other could become a detriment to it.

I did however want to expand on what I did a lot more, there could be a lot more I could add to the experiment such as passing in a Neural network that could see much further into the experiment or properly adding in different types of tiles and rewarding the network appropriately when it moves into them etc. etc. etc. I really feel if I had the time I could add quite a bit more to this project and I felt there were a lot of issues that really slowed down my project. One issue that slowed down my project was that I spent about a month trying to get OpenNN working since I really wanted to work in c++ which didn’t work out very well at all due to OpenNN not having much tutorials or reference material to use to learn how to work with it. This is why I think scrapping everything I had done with OpenNN and C++ and switching over to TenserFlow PyGame and Python was the best decision I could have made for my project as it made developing a Neural network a lot easier and made my project much more successful.

Although the biggest issue that I faced while developing my project were external issues. The first issue was that I fractured my wrist near the start of the academic year and spent collectively about 16 weeks in a cast which massively impacted how efficiently I could work. After I had recovered I then had to completely change my working environment due to the Covid-19 lockdown.

Since I had never worked with Neural networks and TensorFlow before I learned quite a lot from this project and if I were to start this all over I think I would like to increase the scope of the project and include a competing Neural network to the maze with it’s own objective that would interfere with the original Neural network. I would also like to put a lot more work into how I reward my Neural network as I feel I could expand on what I currently have

# References

|  |  |  |
| --- | --- | --- |
| **Referenced Publication** | **Citation** | **Reference** |
| Report | Mastering the game of Go with deep neural networks and tree search | *Silver*, D. (2016). Mastering the game of Go with deep neural networks and tree search. nature |
| Website | (Violante Andre 2019) | Violante , A. (2019, March 19). Simple Reinforcement Learning: Q-learning [Online]. (URL <https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56>). (Accessed may 2020). |