



Darwin Runner

A-LEVEL COMPUTER SCIENCE NON-EXAMINATION ASSESSMENT

Taimur Shaikh | <CANDIDATE NUMBER> | Dubai College | 74615 | 7517/C

Contents

Definitions	5
Key	6
Analysis	6
Background	6
Supervisor.....	7
Analysis Overview	7
Initial Thoughts.....	8
High Level Overview of Proposed Solution Structure.....	8
Background of Problem Concepts.....	9
Meeting 1	12
Predicted Requirements	15
Consideration of Predicted Requirements.....	16
Meeting 2.....	17
Change in Investigation Purpose	19
Evaluation of Learning Algorithms	19
Existing Solutions.....	21
Reflection on Existing Solutions	27
Meeting 3.....	27
Success Criteria	30
Gathering Data for Success Criteria	30
Model User Meeting	31
Meeting 4.....	31
Additional Existing Solution: MarI/O.....	34
Additional Learning Algorithm: NEAT	36
Evaluation of NEAT	37
Consideration of Neural Network Visualisation.....	38
Neural Network Structure and Functionality	38
Final Requirements.....	41
Scope.....	43

Evaluation of Scope.....	44
Modelling	44
Analysis Conclusion	49
Design	50
High Level Overview.....	50
Technologies	54
Overview of User Experience	55
Identifying Core Elements	61
Menus	62
Sensor System	72
Neural Network Structure	79
Activation Functions.....	85
Feed Forward.....	88
Agent	90
Genetic Algorithm	95
Population Initialisation	95
Fitness Function.....	99
Mutation.....	104
Design Conclusion	111
Technical Solution	111
Project Structure	115
Scene Hierarchies.....	115
File Directory.....	117
Prerequisites.....	118
GameManager.cs.....	119
GAManager.cs	121
PlayerMove.cs.....	127
PlayerCollision.cs	131
NN.cs	133
SensorSystem.cs	135

AgentNN.cs	139
CourseManager.cs.....	142
FollowPlayer.cs.....	143
WindowGraph.cs	144
StatManager.cs	148
MainMenu.cs.....	149
OptionsMenu.cs.....	150
GAParamInput.cs.....	150
AudioManager.cs	152
Sound.cs	153
SpawnEnemies.cs	154
HoldValues.cs.....	155
Testing	157
Overview.....	157
Test Plan.....	158
Testing Evidence	169
Success Criteria Results	170
Refinements	170
Evaluation	179
Individual Objectives	179
Overall Effectiveness	183
Analysis of Feedback from Supervisor	184
Further Improvements.....	191
Appendix	193
Sources	195

Definitions

Endless Runner: A sub-genre of platformer games in which the player character is travelling forwards non-stop, and must avoid hazards and/or enemies by jumping, ducking etc. ***Note:** May be shortened to ER in some instances throughout this document.*

Unity: A native cross-platform game engine developed by Unity Technologies. Uses the C# and JavaScript programming languages.

AI agent: An intelligent, autonomous entity that performs available actions in an environment in order to reach some goal state. May use techniques such as knowledge or learning algorithms to choose the optimal action.

Neural Network: An AI model that vaguely mimics the function of a brain using neurons. Can be represented as an undirected, weighted graph. ***Note:** May be shortened to neural net, NN, network, or other abbreviations in some instances throughout this document.*

Genetic Algorithm: An evolutionary algorithm that reflects the process of natural selection where the fittest 'genomes' are selected for reproduction in order to produce offspring (new genomes) of the next generation. ***Note:** May be shortened to GA in some instances throughout this document.*

Q-Learning: A reinforcement learning algorithm that takes actions based on how useful said action is in gaining some future reward or goal.

Ray: Custom Unity data type used to detect collisions

RayCast: Extracted information from Ray.

Abstraction: A method of removing unnecessary details from a problem in order to better understand it.

Sprite: 2D bitmap image integrated into a 2D video game

Tensorflow: Python and JavaScript library developed by Google for scientific computing and machine learning algorithms.

Neuroevolution: The evolution of neural networks using a genetic algorithm.

Markov Decision Process: Model in probability theory for decision making where outcomes are partly random and partly under the control of some decision maker.

OOP: Short for 'Object Oriented Programming'. Programming paradigm that involves the creation of custom 'objects', with the ability to store attributes (fields) and functions (methods).

Encapsulation: The OOP concept of grouping fields with the methods that operate on those fields.

Overfitting: When a model fits too closely to a specific data set and thus becomes worse at generalising for other data sets.

Underfitting: When a model does not fit to data, and thus does not make accurate predictions on any dataset.

Asset Store: A library of game assets for Unity, ranging from textures, animations, models and more.

Inheritance: An OOP technique where an object derives properties and behaviours from a related parent class.

Polymorphism: Giving an action/method one name that is shared up and down a class hierarchy; each class implements said action in a way appropriate to itself

Automatic Garbage Collection: When a language or framework supports the automatic allocation and release of blocks of memory, meaning the programmer does not have to worry about segmentation faults or other memory-related errors.

Key

* : Indicates that the preceding word or phrase is defined in the Keywords section.

* : Indicates that the preceding link or reference can be found in the Sources section.

* : Indicates that the preceding word or phrase will be explained or given context below the paragraph it belongs to.

Pseudocode will be written in the **Hack** font – pseudocode comments will be a gray hue

*All figures that I have created/ captured for the purpose of this project are labelled in order of appearance. **Note that I have not labelled Debug outputs.***

Analysis Background

Dubai College is a British curriculum school in Dubai, United Arab Emirates. I am an A-level student currently enrolled in A-level Computer Science.

Supervisor

Mr. Mark Wood is the Head of Subject for Computer Science at Dubai College and has been since the course started being offered in 2016. He will act as my supervisor for my Investigative Project.

Analysis Overview

This investigation will explore the implementations of Artificial Intelligence in a 3D Endless Runner* environment, and whether I can develop such an AI in my own game that I have already prototyped.

After being briefed on the NEA portion of the course, I detailed my initial thoughts to Mr. Wood. I suggested that I could build upon a 3D endless runner prototype that I worked on during the summer of 2020 using Unity*. My initial intention for this investigation is to implement an AI agent* to oppose the player in some way. I had a few ideas on how to go about this, one being an agent that would, at random intervals, 'challenge' the player with some short, randomly generated task, for example inputting a certain number sequence in a limited amount of time. The player would then have to complete this task before the agent in order to continue running. Shortly after, Mr. Wood gave a fascinating proposal: he wondered whether I could implement an agent, likely using a neural network*, that would essentially be a clone of the player character and run against said character. Moreover, he wondered whether I could train the NN using a genetic algorithm*, and have the player verse a different generation of the network based on their selected difficulty. For instance, the easiest difficulty could see the player running against a relatively early generation that would fail after a short amount of time, whereas the hardest difficulty would likely feature a later generation that can play the game near perfectly.

To implement an effective solution to this proposal, I will conduct further research on existing solutions, as well as determine the specific set of architectures and algorithms that would best suit this investigation. This includes the structure of the neural net as well as the most suitable optimisation algorithm e.g Q-Learning* or genetic algorithm. My current understanding leads me to believe that an NN that evolves with a GA is the best route to take, however this is subject to change depending on the information I find. I will also liaise with Mr. Wood regarding the desired requirements of my project as well as potential additional features.

Initial Thoughts

My current aim is to implement an AI that will provide a suitable challenge to the player whilst playing, whether it be my original idea of 'challenging' the player or Mr. Wood's suggestion, or a completely different type of agent.

The ER prototype that I built during summer was intended to be no more than an experiment to see how much Unity I could learn over the course of a week. As such, it falls victim to poor design and repetitive, sometimes unnecessary code. Before any sort of AI is implemented, a few scripts need to be rewritten cleanly and robustly to prevent any potential setbacks during the Technical Solution phase.

As stated before, I plan to regularly discuss my project with Mr. Wood, but I believe I will also find it useful to interview those who play ERs in their spare time to understand what they look for in terms of playability, difficulty, and rewards. Additionally, perhaps programmers with experience in AI may be able to provide helpful insight.

Mr. Wood's proposal is discussed briefly in Episode 6 of the CS50 Podcast*, so that appears to be a good starting point. The episode gives a comprehensive overview of what AI is, and what the subfield of Machine Learning is within the field of AI.

After having listened to the podcast, I can understand that some of my research should lie in a particular field of machine learning: Reinforcement Learning, as this is used heavily in gaming environments. Currently, I understand the basics of two popular reinforcement learning: Q-learning and genetic algorithms, but I presume this project will require deeper knowledge of them, or perhaps deeper knowledge of an entirely different algorithm.

High Level Overview of Proposed Solution Structure

Although the final premise of my investigation is subject to change, the core concept of implementing an AI to somehow interact with an Endless Runner environment will remain constant. Thus, I can provide a high-level overview of what my implementation will entail.

The process of training the AI may be as such:

- The AI agent(s) will be controlled by a neural network(s) with some sort of numerical information about the current game environment acting as inputs, whose output(s) determine(s) the action taken by whatever the agent(s) is/are controlling.
- Based on these outputs, the relative performance of the NN(s) will be evaluated using some performance metric of my choosing.
- The weight values of the neural net(s) will then be adjusted based on this metric using an optimisation algorithm.

- The process repeats until a suitable solution is found i.e., the AI can achieve the objective I sought it out to do sufficiently well. The criteria for this success depends on what the AI is actually meant to be doing, which has not been finalised.

Background of Problem Concepts

In this section I will provide a detailed background of the key elements that this investigation will likely use. This is to ensure that the premise of my investigation is clear as well as that the techniques implemented are completely unambiguous.

Neural Network:

The main building block of an agent's capacity to learn how to take optimal actions is the neural network that it possesses. A NN is a deep learning model that somewhat mimics the biological neural networks that humans possess in their brains.

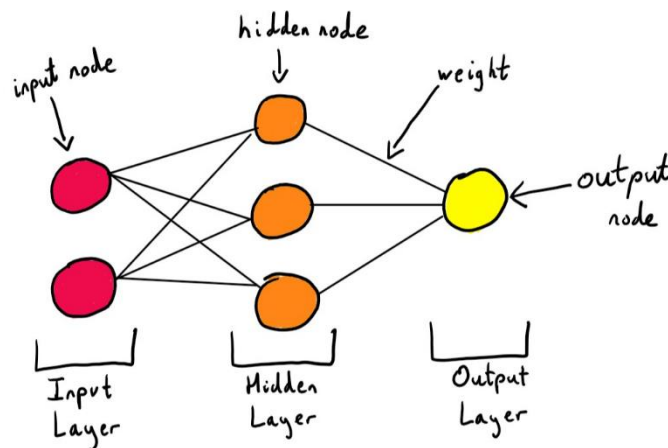


Figure 1: Diagram of Simple Neural Network

The drawing above shows a simple model of an ANN (Artificial Neural Network). Each coloured circle is called a node or a neuron, and each arrow connecting these circles is called an edge or a weight. The arrows all have some numerical weight value associated with them. As you might notice from Figure 1, this representation of a neural network is actually an undirected graph.

This particular structure of neural nets consists of an input layer which contains input nodes, some number of hidden layers which contain hidden nodes, and an output layer which contains output nodes. These, along with other constants, are called the hyperparameters of the network. The network in Figure 1 has one input layer with 3 input nodes, 2 hidden layers, each with 4 nodes, and a singular output node in the output layer.

Each node has a value associated with it. In the case of the inputs, these values are just the inputs themselves. For hidden and output nodes, the value is calculated by taking the **weighted sum** of the calculated values in the previous layer, where the weights are the weight values associated with the arrows. Most neural nets also include some constant in the sum of each layer. This is known as the bias of that particular layer. Normally, this sum is then passed into an **activation function** which transforms the value somehow. This transformed value is called the **activation** of that particular node. So, to compute the output of a neural network given all the weight values and a set of inputs, we simply **feed forward** through the network and calculate the activations for every node until we reach the output layer. The activation of the output node(s) is our final output.

When implementing a neural net in a program, the different layers and weights are normally represented using matrices. Consider the example below which shows how matrices can be used to easily compute the weighted sum of simple two-layer ANN, more commonly known as a Perceptron:

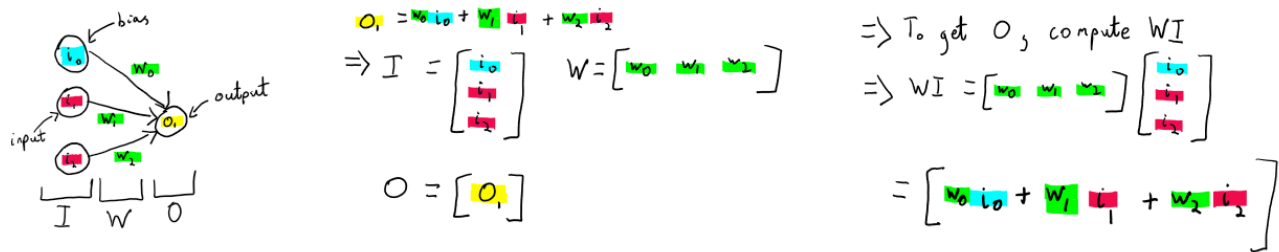


Figure 2: Representation of Perceptron neural net as matrices

The final fundamental concept of neural networks is how it improves. After the inputs are fed forward through the NN, the outputs are passed into a cost function, sometimes called a loss function. This function calculates how 'incorrect' the output was, and thus how good the neural net was at estimating an output. The neural net is then optimised via some technique based on the loss function's output. Optimisation techniques with different learning algorithms will be covered under the next header.

Learning Algorithms:

The feed forward phase of the network is simply a method of computing the output given the network's current set of weights. The way the network actually learns is by altering these weights over time in order to produce more desirable outputs. The weights are altered via a learning algorithm that allows the network to determine the size and direction of the change based on the loss function.

The two most popular learning algorithms used in reinforcement learning (the method of machine learning most commonly used in gaming environments) are Q-Learning and genetic algorithms.

Q-Learning:

This technique revolves around learning to model some function $Q(s, a)$ where s represents a particular state of a search space and a represents an action that can be taken in that state e.g if the state s is a chessboard at the start of a chess match, a viable action a would be to move a white pawn two squares forward. The Q function will take a state-action pair and return the current estimate for the “reward” of taking that action in that state. Using the following formula, we update the Q values for each state-action pair as our agent traverses the search space:

$$Q(s, a) = \text{old estimate} + \alpha(\text{new estimate} - \text{old estimate})$$

Where α is a number between 0 and 1 that determines how much we value the new information. This formula is also known as the Bellman equation.

To implement this technique with neural networks, you simply allow the NN to estimate the Q values for each state action pair in a given environment. Essentially, the NN models its own optimal Q function. This is called deep Q-Learning. The inputs to the network will be states from the environment. For each given state, the network outputs estimated Q values for each action available to take in that state. The objective of this network is to best approximate the optimal Q function. That is to say, the estimated output should be as close to that of the Bellman equation as possible. After the loss function has been calculated, the weights of the neural network are updated via stochastic gradient descent and backpropagation, which are optimisation techniques commonly used for neural nets. This process is repeated until the loss function has been sufficiently minimised.

Genetic Algorithm:

A genetic algorithm is a stochastic search heuristic that loosely mimics the process of Darwinian evolution. It finds an optimal solution to a certain problem by evolving a set (called a population) of different encoded ‘genomes’ (which could be anything: strings, arrays of numbers, neural nets etc.). The genomes are each assigned a ‘fitness’ score based on programmer defined metrics. Then, a subset of the population (usually 2) is selected as ‘parents’. A crossover function is then called on these parents to generate ‘children’, which are also genomes, based on the parents. The parent selection and crossover process can be repeated as many times as needed. The number of repetitions determines the size of the new population of children. Finally, with a programmer defined, usually small probability, each genome can be mutated in some way. The algorithm works by adhering to 3 characteristics that an evolutionary algorithm possesses:

1. **Heredity:** Traits of a parent genome are passed down to their respective child genome.

2. **Natural Selection:** Genomes that are considered fitter individuals are more likely to pass on their traits to children.
3. **Mutation:** Every genome has a set probability of mutating – changing its ‘genotype’ by altering the encoded genome in some way.

By making each individual genome its own neural network, an optimal solution can be found to a learning algorithm through this evolutionary process. Using a GA with neural nets is called neuroevolution. To mutate the NN’s, we can simply change some of the weight values around or employ one of the many other methods that exist today.

Meeting 1

After conducting brief research on different algorithms and solution architectures available to me, I consulted Mr. Wood to get some more ideas and to definitively construct the objectives of my investigation.

Subject: Investigation Purpose and Scope

Duration: 19 minutes

Date: 10/12/2020

Attendees: Me – TS, Mark Wood – MWD

TS: “So Sir, what is it that, going into the project, you’d like me to investigate, and what initial requirements do you think there would be?”

MWD: “Before we go forward, you mentioned that you’ve made the decision to focus on the AI, and that you’ve essentially already made the game. So is the game finished?”

TS: “It definitely needs a little bit of tweaking in order to implement the AI properly.”

MWD: “Okay, but in terms of gameplay, is that something you’re bothered about, or is the gameplay irrelevant? My current understanding of your project is this: You have a game that you intend for people to play, and you want to build a player that is controlled by AI on top of this game, and I can then play the game as a human being and race the AI. Is that correct?”

TS: “More or less, yes. That was your proposal to me after I gave you my initial idea of the AI instead giving short tasks to the player at random intervals whilst the player is running. I am more inclined to go with your proposal, though.”

MWD: “Okay, so for the moment take the AI out of the picture – when I play that game, what’s ready? As in, what’s the gameplay like? Because then, we can talk about what the

AI is actually meant to do. Is it meant to get to the end the fastest, or is it to collect more coins? What kind of approach are you thinking of taking there?"

TS: "Again, the gameplay definitely needs development. I haven't implemented coins yet, but that is something I've been contemplating adding. My idea of how the AI would actually compete against the player, and how its performance and the player's performance would be comparable, is that the player would essentially have to outlive the AI opponent, so to run longer than them. This makes sense since an endless runner is indeed 'endless', so there would be no finish line of sorts, so it's not really a race, but I guess more like a test of endurance."

MWD: "Okay. So what obstacles are there at the moment in the game?"

TS: "Right now there are spikes that the player needs to jump over, and floating objects that they need to slide under, and there are enemies that, in the first version of the game, the player could be shot at by and would need to block, but I'm thinking of taking that feature out in order to reduce scope as the AI may not be able to determine when to block."

MWD: "Are these obstacles dynamic? Are they generated randomly?"

TS: "I wrote a Course Manager script that takes in an array of preset tiles and spawns them in random order as the player runs, whilst also deleting the tiles behind them. I tried to vary these tiles as much as possible in order to prevent any monotony".

MWD: "How is the AI going to know that the obstacles are there?"

TS: "My basic idea regarding that is to give the AI 'sensors' which emit Rays* from the opponent's body. When the Ray collides with another object, you can extract information from that collision using a RayCast*. For example, you can get the distance of the collided object from the opponent, and what the type of the object actually was by reading the object's programmer-defined tag. This information can then be passed as inputs to the AI's neural network. The outputs can be whichever 'action' the AI should take based on those inputs, for example, jump or slide."

MWD: "Right. Sounds good. So, what is the purpose of your project? Is the plan to create an AI that will always beat the player? To beat it most of the time? How are you going to measure and quantify the success of your AI if what it's competing against, in this case, the player, is itself a variable?"

I will now describe the rest of the interview in prose rather than dictate it.

I answered Mr. Wood's question by reminding him of one of his initial suggestions to implement a difficulty select feature which would allow the player to run against different generations of the AI based on which difficulty they selected. The success of the AI can then be quantified by whether it can beat a player of a certain skill level if the game's

difficulty setting is set higher than that skill level. For example, if the player is a beginner, and perhaps not familiar with ERs, they should select the easiest difficulty, where the AI should offer a suitable challenge and be able to beat the player, but still allow the player to win if they play optimally. The medium and highest difficulty should certainly beat this beginner player in every game they try to play against it.

I then realized one drawback of this approach was that the rounds of the game set to an easier difficulty would likely not last as long than a harder difficulty. I then proposed that the AI spawn in at random points **throughout the course** rather than only at the very beginning. This would mitigate the issue as the player would verse the AI multiple times for every difficulty, so there would be no difference in the duration of the run that is directly caused by the game.

Mr. Wood then advised me to be wary of the scope of the game, as too many features may mean that the AI would not be able to learn effectively. For example, he suggested that I not add any power-up items.

We also discussed whether the aesthetics were something that I valued in the game, as well as the target audience for this game. I detailed my intentions of this game being tailored towards any demographic, as is the case for most popular ERs nowadays. Mr. Wood suggested that I research existing ERs and take note of their aesthetic appeal but be sure to evaluate how much of a priority the visuals are to the investigation before making it an objective.

Finally, Mr. Wood and I defined my specific objectives of the investigation. For the time being, we settled that the main focus was to **enhance my pre-existing ER prototype with an AI opponent**. The main distinction I'd like to make with that is that the game itself is not the investigation, but rather the implementation of the AI as well as minor adjustments to the game's source code in order to allow said AI to be added.

Reflection:

This first meeting allowed me to get a better grasp on the technicalities of my investigation, as well as some insight on the scope I should be aiming for. The questions Mr. Wood asked made me consider whether some existing features were better to be removed, and which ones were of higher priority than others. I can firmly conclude that the difficulty select feature is a requirement if the investigation's purpose is to remain as it is now. Another requirement will be the opponent spawning periodically throughout the run rather than at the beginning. As Mr. Wood stated, I need to evaluate whether the visual aspect of the game is something I am going to take into consideration, as well as think about a demographic, even if it is just every age group. Currently, I am content with what the game looks like at the moment, so altering the aesthetics is not of major priority

for this project. If I do visually alter something, however, I will detail what the change was and how I implemented it.

Predicted Requirements

Keeping my first meeting with Mr. Wood in mind, I devised some requirements that I believe I will use for the investigation. These are simply the requirements that I believe should be included in the solution at this point in time and is not final in any way.

In the finished game (after the NN has been successfully trained), the user should be met with these elements:

1. There must be a start menu with at least 3 options, these being:
 - a. A 'Difficulty Select' button that takes the user to a screen where they must choose from Easy, Medium or Hard difficulty. This difficulty section will affect how well the AI opponent is able to perform.
 - b. A 'Settings' button that will take the user to a screen where they can adjust the gameplay parameters (e.g brightness, music and SFX volume) according to their preference.
 - c. A 'Start Game' button that allows the user to start running with the difficulty and settings they selected (Medium difficulty and default settings if they did not alter anything)
 - d. A 'View Training session' button, which takes the user to a visualization of how an agent would be trained from scratch. This includes:
 - i. A population of agents running the course
 - ii. A new population being generated in the next generation
2. The gameplay must be what it is now in the base game, with some added requirements that were unfinished in the current prototype iteration of the game (See 2c, 2e and 2f). These include:
 - a. The player character continually moving forward
 - b. Being able to move left and right
 - c. Jumping a fixed height, and (**Unfinished in prototype**) covering the same amount of forward distance whilst jumping no matter what the player's current running speed is.
 - d. Sliding that allows the player to pass under floating obstacles
 - e. (**Unfinished in prototype**) The player's forward speed must increase at certain distance milestones so as to increase the difficulty of the run.
 - f. (**Unfinished in prototype**) Coins must be placed along the course in such a way that the player is able to collect all the coins in a single line if they play optimally. The player must be able to see their coin count in-game.

3. The player must be able to view their high score (highest distance travelled) at all times whilst running and must be notified when they pass it during any given run via a visual and/or audio cue.

Consideration of Predicted Requirements

Upon evaluation of these devised requirements as well as my meeting with Mr. Wood, I have started to notice a few flaws and potential risks in my proposed investigation. For instance, my solution to Mr. Wood's question about the AI's performance being quantifiable (See [Meeting 1](#)) is not inherently quantitative; it appears that there is no sound method to verify that the player is a beginner or novice in ERs, and it is difficult to determine whether the AI would perform well enough for a certain difficulty, as it would have to beat players of that skill level some fraction of the number of times played, but working this fraction out becomes rather ambiguous and subjective.

That being said, after conducting some research, I found a research paper* by students at the Conservatoire National des Arts et Métiers in Paris concerning an investigation on how to calculate the level of difficulty using in-game parameters. The paper attempts to find some sound mathematical formula to calculate the player's skill level and the difficulty of the game as it relates to said skill level and the current circumstances of the game environment e.g the level number. Whilst promising, the researcher's investigation was in its infancy, and no tests on human players were done to verify their hypotheses, so following their method may be prone to a flawed solution.

An alternate pathway to take in implementing this AI is to disregard the player entirely and just focus on implementing an AI that can play an ER perfectly, and instead allow the user to view a visualization of the agent learning to run rather than playing the ER themselves. This would be equivalent to a more developed version of [initial requirement 1d](#). With this method, objectives can be laid out much more easily, and success of the investigation is strictly quantifiable by measuring the duration that the AI can survive when running the course. Contrarily, it would fail to showcase the practicality of an AI agent playing against a human, which is normally a main goal when implementing AI in gaming environments.

Before making any definitive changes to my investigation, I will consult Mr. Stuart Forsyth, another one of my Computer Science teachers, regarding the technical complexities of this new approach as well as its relative scope.

Mr. Forsyth is familiar with machine learning algorithms, and has recently completed Master's in Artificial Intelligence, so his thoughts on my proposed solution requirements will prove useful.

Meeting 2

This is the brief but insightful conversation I had with Mr. Forsyth over a Microsoft Teams call.

Subject: Potential Change in Investigation purpose and Technical Complexities

Duration: 6 minutes

Date: 7/1/2021

Attendees: Me – TS, Stuart Forsyth – SFO

TS: “Thanks for meeting with me today, Sir. So Mr. Wood recommended that, if I were to slightly change the purpose of my investigation, I should consult you, as we could delve deeper into the technical aspects of it as well as the relative scope.”

I then recounted my initial investigation purpose of enhancing an ER with an AI opponent and then voiced my concerns to Mr. Forsyth regarding the potential flaws in my original investigation premise (which are listed in the [Consideration of Requirements](#) section) as well as my proposed alternate implementation of disregarding the player entirely. Mr. Forsyth agreed that the issues I brought up pose a threat to the measurability of the success of my project and stated that he would be much more comfortable with my new, more focused approach.

TS: “In terms of the scope of this project, how far would you say I should go with this AI? What is the upper bound of what it should be able to do?”

SFO: “Well, did you see Tom’s game yesterday? That would have been a phenomenal project, actually.”

TS: “Right. It was a really cool project. That’s actually similar to what I’m proposing, as it’s a genetic algorithm evolving a population of neural networks to a play a game, so that’s good to hear that that would be a good project.”

SFO: “So your aims should be roughly of that scope and size.”

TS: “Okay, so I should have something of similar scale to that if I were to do a genetic approach? A base game, an evolving population of agents, each with a neural network as a brain, and a method of graphing fitness over time as well? I hadn’t actually thought of implementing a graph of fitness but having said that I think that would be really useful in displaying performance of the algorithm.”

SFO: “You definitely need a graph to show the improvement. And you need to show other statistics about how well it does, and detail how it actually learns, but you don’t need to solve it the same way as Tom did. You can have a completely different approach.”

TS: "Right. It just so happens that I was looking over which optimisation algorithm to use for my project, and I was going back and forth between a genetic algorithm and Q-learning, but I'm now more set on the genetic algorithm just as Tom used."

SFO: "That's upto you. Try one out, and if you're not happy with the result, you could potentially re-engineer your solution for the other algorithm. Make sure to discuss that in your analysis."

TS: "For sure. So, in terms of the technical complexities of my project. Where exactly would you say those lie? Obviously one of them will be matrix operations from working with a neural network."

SFO: "One that I see immediately is the complex data structure that is the network itself. It would be much more complex than a hash table or what have you. There are also the other mathematical algorithms that you will use to manipulate it. That's assuming you will write the network from scratch and not just use a library to implement the whole thing. If you write it from scratch, you've got at least three high complexity components in your project."

TS: "Okay. And just to clarify, Tom was showing me a small matrix library he made for his game, but that's not something I necessarily need to do, right?"

SFO: "No. I'd say you won't need to. It'd be a waste of time, actually."

TS: "Okay. Because I believe there is a very well optimised maths library in C# that has all the matrix operations I need."

SFO: "Sounds good."

TS: "Thanks for meeting with me Sir."

SFO: "No problem."

** : The day before this meeting, Tom Rietjens, one of my peers, presented a Flappy Bird* AI during our weekly Computer Science and Robotics Society meeting. He created this program over the 2020 winter holidays.*

Reflection:

After Mr. Forsyth's approval of my new investigation premise, I am more or less set on it. So, rather than enhancing an ER with an AI opponent, I will instead **implement an AI that can play an endless runner game**. With this investigation goal, success can easily be measured as how well the AI performs under my pre-defined success conditions. This would most likely be the distance that the AI is able to run without dying but may include things like how many coins it collects along the way as well.

Mr. Forsyth made sure to emphasise the importance of having a graph feature that allows the user to view a plot of the agent(s)' fitness over time. Moreover, he mentioned 'other statistics', which likely refer to other utilities involving the current run of the algorithm, for instance, the maximum fitness any one agent has achieved, as well as the average fitness across all generations so far. This would also divulge further information as to the current performance of the AI.

I am confident in saying that this implementation will include several Band A technical complexities. This includes a complex data model in the form of the neural network, which will likely be written as a C# object, the complex matrix operations within the feed forward stage of the NN, as well as the other mathematical algorithms used in the GA phase such as a crossover function*.

Although I am leaning more heavily towards a genetic approach, I shall also evaluate the projected effectiveness of an approach that uses Q-Learning instead, as it may have more benefits than the genetic approach for this particular implementation.

Change in Investigation Purpose

The premise of this investigation has now officially changed to: **To create a visualisation of an AI that can play a 3D Endless Runner.**

Because of this change in premise, adjustments need to be made regarding recipients of the investigation as well as requirements. The target audience is no longer those who casually play ERs, but instead those who wish to **understand and investigate how an AI may learn how to traverse a 3D ER environment**. I will solidify this through the use of helpful utilities and a graph plot of performance over time in order to clearly and effectively depict information about the project. After I have performed a more comprehensive evaluation of all the elements that will go into this kind of implementation, I will devise some final objectives.

Evaluation of Learning Algorithms

The two learning algorithms that are most commonly used in reinforcement learning based AI implementations in gaming are Q-Learning and neuroevolution* using a genetic algorithm.

Q-Learning:

Benefits:

- Q-learning would allow for only one neural net to be initialised at runtime, as there is no need for a population of agents as Q-learning is not an evolutionary algorithm. This may improve the framerate of the game as well as save memory.
- Q-learning will require less individual, self-contained scripts to be written as the processes of feeding forward the NN, computing the reward value and modifying the network via backpropagation are likely more coherent when placed in one to three scripts only, whereas a genetic approach would contain multiple scripts for the initialisation of the network and feeding forward alongside another set of scripts to run an evolution on a population of networks using the GA.
- Q-Learning can be clearly and unambiguously modelled using a Markov Decision Process*. A reinforcement learning based genetic approach is less well defined than this, and thus may be prone to more bugs early on.

Neuroevolution with a genetic algorithm:

Benefits:

- The NNs being evolved by a GA are less likely to fall into local minima/maxima than Q-Learning. This is due to the algorithm searching parallel from a population of genomes rather than a single point. This means that they are most likely to find the most optimal set of weights of the network (the global minimum/maximum).
- A GA will not require backpropagation, which would be computationally expensive. Other instances of avoiding expensive calculations are the use of a fitness score which does not rely on derivatives or any other auxiliary information.
- A GA is both easier to understand and easier to implement than the Q-Learning algorithm, as it does not involve any calculus operations such as partial derivatives.

Final Evaluation:

I will employ a genetic algorithm for this project. This is mainly due to the fact that it is less likely to fall into local optima than traditional learning algorithms. Local optima occurs when the optimisation technique has reached a solution that is optimal as compared to the solutions that are 'close' to it. This is equivalent to a local minimum/maximum on a graph plot of the model's performance metric over its output. The following example depicts this for some function output W plotted against $J(W)$, representing the cost function of this output, which measures how bad the current parameters of the W function are.



Figure 3: Simplified graph of the cost function of output against output itself

Although the framerate may be impacted when evolving the population, I will attempt to perform optimisations once a prototype of the AI is complete in order to improve performance. This may include reducing post-processing effects and using lower resolution textures. I believe the whole investigation will benefit from the choice of a GA as opposed to Q-Learning, as the concepts and algorithms involved in an evolutionary algorithm are simple to explain and demonstrate coherently, whereas a Q-Learning approach would require more strictly numerical analysis that, whilst very rigorously quantifiable, may become cryptic and ambiguous over time.

I will likely need to devise my own techniques and algorithms specific to this implementation, as there is no definitive method of implementing evolutionary algorithms with NNs; Q-Learning has a more detailed, standardised process, but I still believe a genetic approach will produce a more streamlined, less convoluted solution.

Existing Solutions

It will certainly prove useful to look at existing implementations of AI in an ER environment, especially those that make use of GA as I am leaning towards a neuroevolutionary approach.

1. endless-runner repository, “Neuroevolution AI solving Endless Runner game” – GitHub User LozzLol *

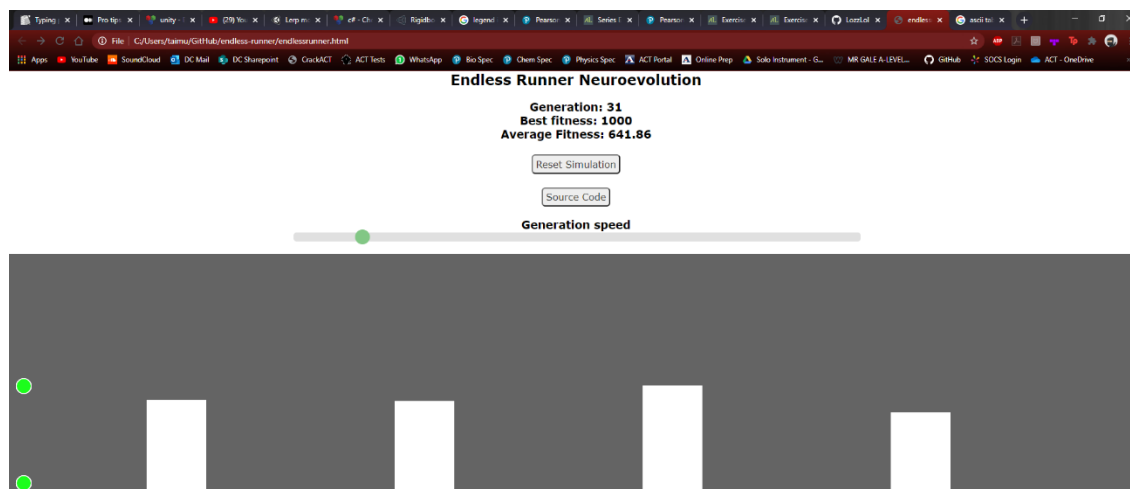


Figure 4: Screenshot from Solution 1

Description:

This repository contains a simple browser-based ER neuroevolution visualization written using the p5 JavaScript library with one obstacle type of varying heights, and the only action the agents can perform is jump. Each agent, represented by a green ball, is controlled by a neural net which evolves using a genetic algorithm. Detailed above the window displaying the agents and obstacles are helpful utilities about the performance of the NN, such as the current generation, best fitness of any genome and the average fitness across all genomes. The program also allows the user to alter the speed at which the runner plays, thus allowing the NNs to evolve quicker. After an average fitness of about 640, most of the genomes in the gene pool of one generation were able to perform nearly perfectly.

What I like about the solution:

This interface is simply laid out and provides an abstraction* of how a genetic algorithm would go about evolving a neural net by not displaying any technical complexities. Moreover, the utilities provide the user with quantitative information to determine the success of the agents at that point in time (by checking the agent's max and average fitness, the user can see the best the agent has ever performed as well as how well it is generally performing, respectively), which would be helpful in my implementation. The simulation allows the user to see all genomes in a generation at once rather than seeing them all sequentially, which gives a better sense of the GA representing evolution within a species of creatures. As generations progress, one can then see the average performance of all the genomes improve. This allows a user to better visualize how the GA is working, and how it mimics the real process of natural selection and evolution.

What I would do differently:

The gameplay is rather stripped down and simple compared to ERs. If I were to visualize the evolution of an ER agent, I would allow the agent to take more actions in the environment instead of just jumping. Consequently, I would also incorporate more obstacles that are conducive to these actions. For instance, rather than only having pillars that protrude upwards from the ground that the agent needs to jump over, I would also have ones that protrude downwards from the ceiling that they would need to slide under.

In addition, I would make the game more visually appealing by adding sprites* rather than just simple shapes. Whilst the minimalist look is somewhat enticing, it makes for a rather bland visual experience, and so the user may lose focus of what is happening. The utilities do mitigate this issue, though.

2. GeneticAlgorithms repository, “Automatic Chrome Dino Game” – GitHub User aayusharora*

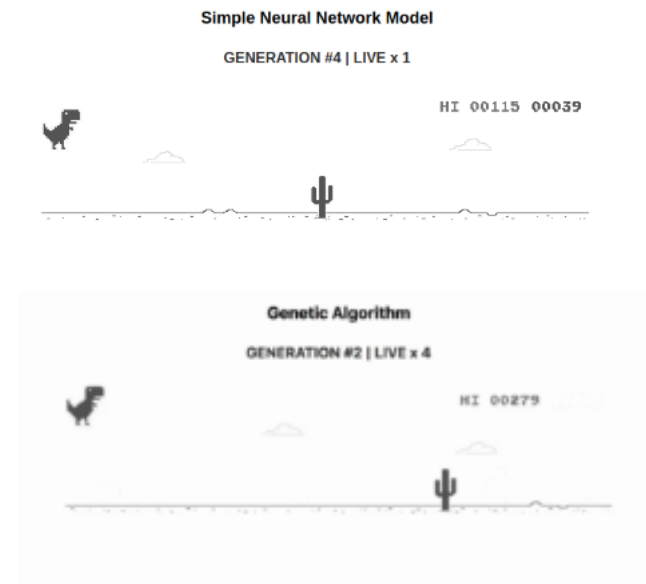


Figure 5: Screenshots from Solution 2

Description:

This repository houses two approaches to creating an AI that can play the popular Google Chrome Dino ER Game, which does not have a specific link, but rather can be played if the device intended for it to be played on encounters any issues with Internet connection. The AI strives to play the game just as a human would. In the game, the agent is a neural network that controls a 2D dinosaur which jumps over cacti and ducks under birds. As such, the agent can perform these two actions only. The first approach in this repository uses the Tensorflow* library in JavaScript to implement a neural network that learns to play optimally. The second does the same but instead does not use Tensorflow, but rather evolves the co-efficients to an equation that determines the agent's next move using a genetic algorithm. Similar to the previous existing solution, utilities are listed above that detail information about the visualisation. The Medium articles* that accompany this repository alongside the source code itself provide more information regarding the technicalities of the project. For instance, the 3 inputs to the neural net in the first approach are the same 3 values that appear in the genetic algorithm equation approach, whose co-efficients the algorithm is attempting to optimise. These 3 values are the speed of the game, the width of the oncoming obstacle and the distance of said obstacle from the dinosaur.

What I like about the solution:

Displaying two different AI approaches to reach a near optimal solution in a gaming environment is useful in determining which approach is most suitable in that specific

context by evaluating the success of each approach. For example, in nearly all of my runs of the two programs, the genetic algorithm approach was able to reach a stage where it could play near perfectly (That is to say, it was able to not die for 5 or more minutes) far faster than the neural network approach. This means that, in this specific application, the genetic approach is more optimal.

The utilities present are clearly laid out and easy to understand, allowing the user to better understand the visualisation's current performance. Other than that, the user interface is near identical to that of the original Chrome Dino game, with score and high score in the top left of the window.

Like my own proposed investigation, this solution implements AI in a pre-existing game which does not use any AI. Thus, the objectives and requirements of this project are likely similar to mine, and so the code where the programmer modified the Dino object already coded by Google to allow for an agent to take the actions that a human player normally would may serve as a helpful example during my Technical Solution.

What I would do differently:

The existing utilities, whilst helpful, can certainly be built upon. Currently, in both approaches, the only information displayed is the generation that the program is currently on (in approach 1, it is not a "generation" per se, as there is only a single neural net that is adjusting its own weights rather than the weights being crossed over with other NNs and passed on the next set of networks) and the "life" of the currently displayed agent. I would take the extra step in displaying the maximum fitness that any agent has achieved in a particular run of the program, as well as the average fitness. Finally, I would draw some sort of graph plot on the screen of fitness over time. This would provide further information in regard to how quickly and successfully the program has achieved its goal.

3. "Endless Runner Neural Network", Liam Patel *

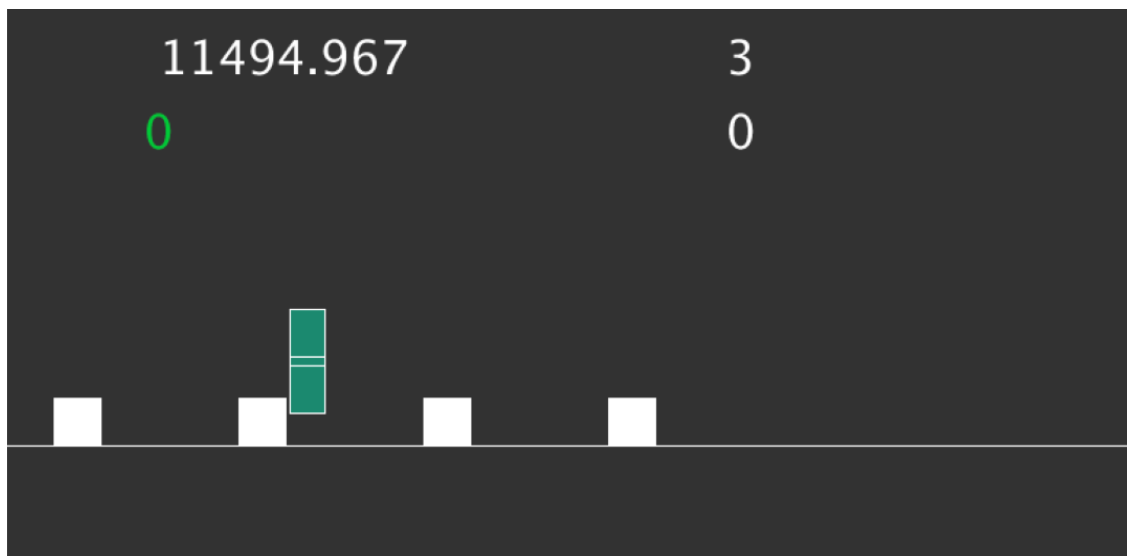


Figure 6: Screenshot from Solution 3

Description:

This side-scrolling Endless Runner is built in the Processing mode of Python. The author of this article and project describes that they made this program in order to 'learn more about both genetic algorithms and the creation of neural networks'. The whole article provides an overview of each of the principal components of the code, for example the creation of the player and neural net class, the initialisation of the network, and the different subroutines of the genetic algorithm that is used to adapt the network.

The agent controls the blue rectangle shown above, which is kept horizontally stationary whilst the obstacles (white boxes) move towards it. This is a common technique in ERs; rather than the player moving, the environment moves towards the player. Jumping is controlled by hitting the spacebar, which modifies a vector value to represent the vertical movement of the rectangle. The fitness function of the genetic algorithm takes into account the distance travelled as well as the number of white boxes jumped over.

The network is made completely from scratch, with no scientific computing libraries such as Numpy or Scikit Learn due to these not being compatible with Processing. This means that there were no matrix libraries available to the project's creator. As a workaround, they created a new Node class, which represented a singular node in a NN. The Neural Network class then initialised its main fields as lists of Node objects, the quantities of which are determined the class parameters. This constructor of the Neural Network class is as such:

```
class NN:
    def __init__(self,numIn,numOut,setWeights):
        self.inp = [N('inp') for i in range(numIn)]
        self.out = [N('out') for i in range(numOut)]
        self.hid = [N('hid'), N('hid'),N('hid')]
        self.setWeights = setWeights
        self.active = True
        self.score = 0
        self.fitness = 0.0
```

Figure 7: Code snippet from the N() class of Solution 3

Where N() is the Node class.

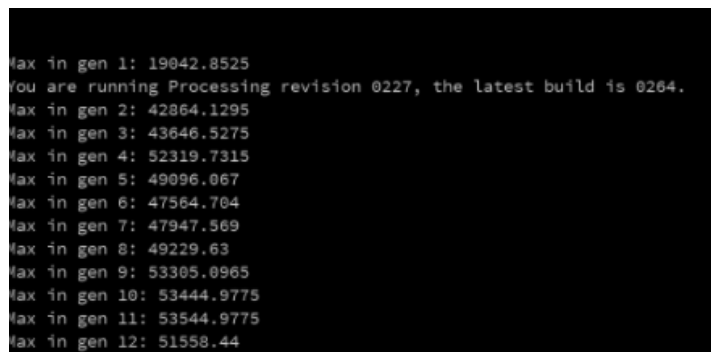
What I like about the solution:

The project follows a strict OOP* paradigm, with all processes contained within a small number of Python classes. This fulfills the ideology of Encapsulation*, and so this code is highly reusable. It can likely be easily migrated to another gaming environment and, with the movement system as well as the GA's fitness function modified, the objectives of the

agent can be completely different. For example, rather than attempting to get as far as possible in an ER, the agent could instead aim to complete a level as fast as possible in a more traditional 2D platformer.

The creation of a separate Node class to solve the issue of not being able to use matrices to represent the nodes is a clever fix to an otherwise troublesome issue. In a sense, Patel is creating their own matrix through the use of a 2D array to house all the networks' internal weight values.

The creator also clearly shows certain final results of the project. In other words, the performance of the AI can clearly be analysed as it is strictly numerical. The command terminal upon a run of the program appears as such:



```
Max in gen 1: 19042.8525
You are running Processing revision 0227, the latest build is 0264.
Max in gen 2: 42864.1295
Max in gen 3: 43646.5275
Max in gen 4: 52319.7315
Max in gen 5: 49096.067
Max in gen 6: 47564.704
Max in gen 7: 47947.569
Max in gen 8: 49229.63
Max in gen 9: 53305.0965
Max in gen 10: 53444.9775
Max in gen 11: 53544.9775
Max in gen 12: 51558.44
```

Figure 7: Terminal output from Solution 3

Figure 7 shows the max fitnesses for each generation of the evolution up until generation 12. As is shown by the results, the program is able to incrementally improve the performance of the neural nets. This outcome is clearly shown and is completely quantifiable.

What I would do differently:

The utilities that are drawn on the game screen do not have any headings to denote what the utility actually is. This can cause some ambiguity for users who are not paying attention to the visualisation from the beginning. Moreover, the metrics displayed could be improved by displaying the overall max fitness and average fitness rather than just the current fitness.

The code is not commented, and so no explanation of what different parts of the program do exist. For techniques as advanced as neural networks and evolutionary algorithms, an absence of comments means that someone without experience using them will be easily confused. There is a detailed explanation of the program in the article, but having this information included within the program itself is more conducive to future debugging and improvements.

Like the first existing solution, this program could benefit from more diverse gameplay. That is to say, introducing more actions for the agent to perform e.g sliding under things instead of only jumping over equally sized boxes. Granted, this may be more work for the GA, but it would make for a more versatile agent that may be more easily ported to other gaming contexts.

Reflection on Existing Solutions

These solutions all have their own individual merits that I can certainly benefit from if I incorporate a similar technique/feature. For instance, the straightforward layout of Solution 1's utilities would enhance the practicality of my project, as the user will always be able to clearly see the information required to understand the evolution and performance of the agents. Moreover, the two-solution approach can be adapted for my specific desires e.g I could have two different populations of agents with two different fitness functions; one that favours how far the agent has travelled, and another that instead favours how many coins the agent has collected. Thus, a potential objective can be to visualize how different objectives affect the agents' behaviour in their environment.

Meeting 3

I decided to consult Mr. Wood again to propose my new investigation premise and listen to any concerns or suggestions he has that I may not have thought of initially. This may alter my investigation even further.

Subject: Verification of Investigation premise and devising Final Requirements

Duration: 14 minutes

Date: 19/1/2021

Attendees: Me – TS, Mark Wood – MWD

TS: "Hi Sir, I just wanted to have this meeting to say that I ironed out my investigation with Mr. Forsyth. So rather than go with the player approach where I pit the AI against the player, I instead am planning on making a visualisation of agents evolving by themselves. I'll put utilities and graphs on the screen to show a realtime evaluation, such as max fitness over time."

MWD: "Okay. So could you describe to me what it would look like when you run it?"

TS: "So, since it's a genetic algorithm, you'll most likely see multiple agents running at the same time, and since all of their neural networks are slightly different, you might see them act differently, like some may move side to side or jump more often than others. And then

you'll also see different metrics displayed such as max fitness, as in the best an agent's done so far, as well as a graph of fitness over time, so you can see how the average fitness has improved over time."

MWD: "I'm liking all of that at the minute. Is it going to be interactive? Or am I just going to watch it? What I mean by that is, can I, as a user, plug in any values to manipulate the process, or is it simply that I'm just viewing what you've made."

TS: "One thing I could look into is a slider for speed, as in how fast the evolution is actually going in terms of timesteps per second, so that a user could reduce the amount of time they are watching the visualisation in case they don't want to sit through it for an extended period. And also, maybe another parameter that the user could control is the mutation rate of the genetic algorithm, which is just a probability of changing an agent in some way in a given generation. It's a lot like emulating how real organisms mutate over time. Maybe I could even allow the user to define the population size, up to a certain threshold, of course."

MWD: "I can see the complexities in your project, and I know you've discussed with Mr. Forsyth. Without a doubt, the technical complexities are there. I just need to get two things in my head. First, what are you going to test?"

TS: "Well, one of the reasons I actually switched my investigation purpose was because the player approach would be a lot harder to test and not be as strictly quantifiable. With this new approach, with the AI alone, rather than testing how well it does against the player, which is subjective in itself, we can see how well the AI has actually evolved, like how far it's gone. I feel like that will be the main factor I will assess."

MWD: "Alright. But what are you *literally* going to test? For example, there's a past project that has something like 57 different tests."

TS: "One of them could be testing the handling of the input that we mentioned before, like the mutation rate and population size."

MWD: "Here's the thing: in terms of testing, I've never had a case where every test wasn't in the order 'Input, Process, Output', like for those user defined inputs you just mentioned. That isn't necessarily a bad thing, it's just new. In the majority of your tests, you wouldn't know the inputs as it's all automated, however I have an abstracted view of the processes, and the outputs are displayed to you. I may be wrong though, so describe to me what you think you're going to test."

TS: "Maybe what I could do is something like this. Basically, there are a few steps that need to run in a genetic algorithm, each of which can be thought of as a self-contained module. For example, there's a crossover phase and a mutation phase that happen separately. I could test all these different modules with sample values that the functions will get and expect a certain output, although I won't know the exact values they'll get

during actual runtime of the AI as an evolutionary process is at least slightly different in every run.”

MWD: “Right. And can you do that in a modular fashion? Or does it have to be tested holllistically, with everything put together?”

TS: “You can do it modularly. You’d just have to give each module different parameters as they’d likely take in slightly modified versions of the data from the previous modules.”

MWD: “Okay. So now let’s talk about the investigative aspect. You’ve changed your objective, so now what are you investigating?”

TS: “It’s how well I can make an AI perform in an Endless Runner, which evolves using a genetic algorithm”.

I will now describe the rest of the interview in prose rather than dictate it.

Mr. Wood then gave me some on context on existing projects that are similar to mine, in that employ some form of AI solution. He made the distinction that these projects have clear, measurable success criteria. For instance, Mr. Forsyth has developed an AI that marks short answer A-Level Computer Science questions for his master’s project. His success criteria were whether his system could correctly mark 50% of his dataset of sample answers with an accuracy of 90%. He asked me how I would apply this principle to my investigation. I explained that one potential metric to evaluate success on could simply be the duration that the agents can run for after my evolution stage is done e.g if the agents can run for 10 minutes or more, I would deem the investigation successful.

Mr. Wood then proposed an interesting idea. Since I will be building this on top of my existing ER game, he suggested that one of my overarching objectives **to develop the AI such that it can play as good as a human or better**. What this entails is that I would take a sample of people and time how long they can last in my ER. Then, after having built the AI, run the evolution up until a certain point, then test the agents’ performance against the data I collected on the human players. Mr. Wood believed that this would prove for a more focused, quantifiable Testing phase, as well as a clearer investigation. It wouldn’t actually change what I’m doing, but rather the end criteria are slightly different.

Reflection:

More and more details about the potential structure and intricacies of my investigation are becoming clear to me now. It would certainly enhance the value of the project if I allowed the user to define their own values for some of the parameters of the genetic algorithm. I believe it would allow them to experiment with these values and see how altering them affects the evolutionary process, allowing them to further understand it.

Mr. Wood’s suggestion about building the AI with the main objective being for it to play as well as a human (or better) is definitely something that would more starkly highlight

the success of my project. Moreover, the method of gathering data for this objective is very straightforward; I could ask a handful of my peers to try their hand at the base game, and I could record their average duration of survival or just their score over a set number of runs.

I will solidify my ideas on this suggestion and then state the requirements for this investigation after having analysed and fully scoped it as seen in this section.

Success Criteria

My success criteria will be, as Mr. Wood proposed, whether my AI system can play at the same level or higher than any human player barring myself (as I developed the game, so I hold more detailed knowledge than any other player in a sample of people that I will select, thus the results would not be valid if I was included in the sample).

The base game is ready for this kind of testing. That is to say, it has all the characteristics of a typical endless runner, and so any player would be able to play without much difficulty.

One of the criteria for the success of my investigation will be to test whether my AI can **play as good as or better than a human player**.

Thus, the main criteria for success I have is as follows:

After sufficient training, at least one AI agent that has been generated through my genetic algorithm should be able to obtain the same or higher score as the best performing human that I examined. Beforehand, I will take a sample of my peers and ask them to play my existing ER game for a fixed number of runs. I will record their scores and take the average of their scores across all runs, excluding anomalous results which may be due to misclicks or technical issues.

I must be able to create an AI that can reach a sufficient maximum performance such that the user can easily see and gauge how much it has improved over the course of that run. This will be detailed further in the sections following my Analysis.

Gathering Data for Success Criteria

For my Success Criteria, I will compile a set of results from 8 of my peers. They will play my base ER game for 5 runs. I will then take the average of their scores. The highest of these averages will be the benchmark for my AI – if it achieves a score greater than or equal to that score, then it has succeeded in the first criterion.

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College.
Centre Number: 74615. Component Code: 7517/C.

A table of the scores of the 8 human players is shown in Figure 8:

Student	Run 1	Run 2	Run 3	Run 4	Run 5	Average
1. Aman	121	29	116	72	80	83.6
2. Tom	100	100	48	74	82	80.8
3. Pratyush	47	162	54	100	111	106.75
4. Jafar	30	123	52	57	44	61.2
5. Shrish	100	100	123	50	111	108.5
6. Ping	62	75	132	54	34	71.4
7. Sam	56	54	33	23	50	43.2
8. Nik	20	23	32	23	43	28.2

Figure 8: Table of Results of Scores

NOTE: The values highlighted in red are anomalous and were due to some form of error during that run; they were not considered in that person's average score.

From the table we can see that, on average, Student 5 performed the best with a score of 108.5. This translates to about 28 seconds of playing time. So, **if at least one of my generated agents last longer than 28 seconds, the criteria have been achieved.**

Model User Meeting

In this section I will interview one of my peers, Aman Doshi. He is also a first year A-Level CS student. He has specified to me that he is not familiar with genetic algorithms or other evolutionary algorithms for that matter. As a result, I thought it would be useful to ask him about what features he would like to see, as I intend for the project to be accessible to people inexperienced in the inner workings of it. I decided to meet with Aman to discuss what he would like to see from my solution.

Meeting 4

This first meeting was with Aman over a Microsoft Teams call.

Subject: Solution user desires

Duration: 8 minutes

Date: 22/1/2021

Attendees: Me – TS, Aman Doshi – AD

TS: "So, Aman, just as a preface, how familiar would you say are with neural networks and genetic algorithms?"

AD: "I'm not really familiar with them at all."

TS: "Okay. So, you don't know the steps of neuroevolution using a genetic algorithm."

AD: "No, I don't."

TS: "Okay. So, what features would you want to see in a visualisation of a genetic algorithm so that you can learn the most about how they work?"

AD: "From what I understand at this point, the weights of the nodes change after every generation, right?"

TS: "Sort of. In every generation, new neural networks are generated based on the ones in the previous generation."

AD: "Right. What I think would be kind of cool is seeing the changes between each generation as you watch the visualisation happen. So, whilst watching the runner in your case, I think it'd be really insightful to see how the weights compare between different runs of the program and seeing the similarities."

TS: "Right, I see. So how do you think that information could be visually expressed in the program?"

AD: "My best guess would be somehow seeing which nodes of the network change the most, or maybe the average change in the nodes. Maybe as you see the relative changes in those nodes start to decrease, you can sort of visualise the network refining itself and becoming really good, which I think would help understand the process."

TS: "I see. Would you want to see an actual visual representation of the network itself, like in the weighted graph format they are usually represented in? Or would you prefer some other kind of representation?"

AD: "Ideally, the graph format, as they seem to be the most simplified view of them. But I don't know how that may affect how your game may run. Because I presume there's a lot of processing power that comes with rendering that dynamic information every frame. It could be possible though. It's just that I'm doing visualisations as well*, so I know it may get difficult to display a graph on the screen if you're not using an external library, which you could do. It's probably better to do that, actually."

TS: "I see what you mean. These are all just ideas at this point, so none of these are final. How would you feel about having a chart that plots average fitness of the neural networks over time?"

AD: “Definitely would be good. I think that would really simplify the process. Just one question. How do you think you’re going to show the changes in weights between the layers of the network? Maybe you could represent them using different colors of the edges.”

TS: “I was just thinking that. Have you seen that Mario AI by the Youtuber SethBling*? For his visualisation he did something like that, where he actually drew the neural network at the top of the game view, and he used colours for changes in weights over generations.”

AD: “I haven’t seen the Mario AI, but that idea seems really good.”

TS: “And like what you said before; if there’s a big change from one generation to the next, that change could be visually represented by a lot of the edges changing colours.”

AD: “Right. That sounds really interesting.”

TS: “One last thing. In order to make the program somewhat interactive, I’m thinking of allowing the user to input a handful of parameters of the genetic algorithm at the beginning of each run. These include the mutation rate and population size, but it doesn’t really matter what they are, because I want the user to play around with them and see how they affect the performance of that particular run.”

AD: “Yeah, definitely. Seeing how a change in the parameters makes a change in the neural network would be really cool. I guess it wouldn’t teach you the intricacies of how it works, but it would help you understand the importance of each parameter that the user is allowed to define.”

TS: “Sounds good to me. Thanks Aman.”

AD: “No problem.”

** This project will be elaborate upon in the following sections.*

** Aman’s NEA is a web-based project revolving around different data structures and visualisation of algorithms on them e.g Depth First Search.*

Reflection:

The main thing I took away from this meeting was Aman’s suggestion of a visual representation of the neural network and it’s changes each generation. This immediately reminded me of MarI/O, which I had otherwise forgotten about. It’s a program written by the Internet personality SethBling. The program is of a similar ethos to mine: it uses a genetic algorithm to evolve a neural network to play a level of Super Mario World, a game for the Super Nintendo Entertainment System (SNES). In it, a visualisation of the network used to control the Mario is shown at the top of the screen, and changes in the parameters

of the network between generations is shown via colour coding the connections that correspond to those weights. This is very similar to what Aman was suggesting. One main difference between my proposed solution and MarI/O is that MarI/O uses a variant genetic algorithm called NEAT (Neuroevolution of augmenting topologies). MarI/O is such a prime example of a neuroevolution implementation in a gaming context that I should examine it as an existing solution in the following sections. Additionally, I should dedicate some time to researching NEAT as it is a viable option for my implementation, however I am not too familiar with it, so it would have to offer immediately tangible benefits for me to switch my learning algorithm to it instead of my base genetic algorithm (the algorithm that will evolve neural networks of fixed topologies).

Aman did express concerns about how visualising the networks may affect the runtime of my game, and how perhaps using an external library which is optimised to render them may streamline the process. I will look into it and see if there are Unity compatible C# libraries that achieve something like this.

This meeting as a whole got to me consider what features I would need in a finished implementation to marry my two [Success Criteria](#) together into one cohesive implementation. I must create the visualisation such that it is effective enough to generate an agent that can run as well as a human

Additional Existing Solution: MarI/O

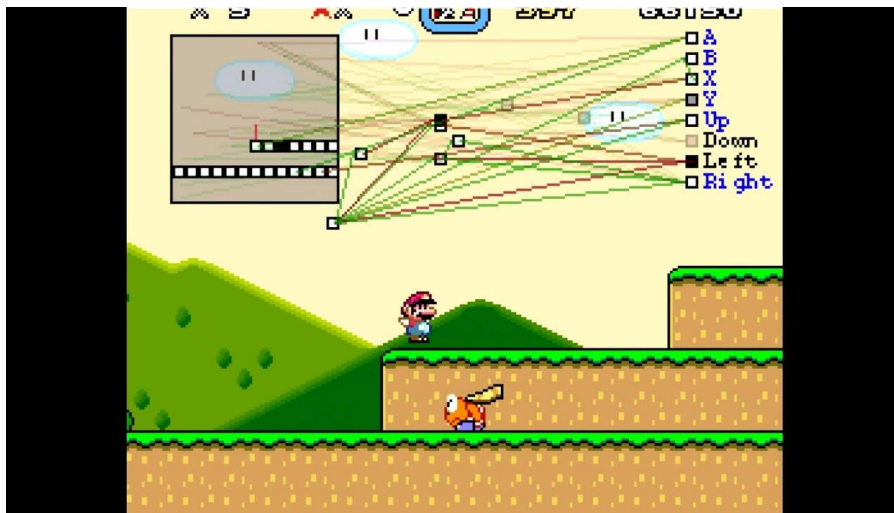


Figure 9: Screenshot from MarI/O

Description:

This is a neuroevolution program that evolves an AI agent to complete the level 'Donut Plains 1' from Super Mario World in the fastest time possible. The creator of this program, SethBling, has uploaded a YouTube video* providing more information about the project as well giving a brief overview of how it works.

The input that the neural network that controls the character receives is shown in the top left region of the above image. It's a simplified view of the game window, where the red line represents the player character, white blocks represent where the player can safely stand, and black blocks represent moving objects such as powerups and enemies. The outputs of the network are the buttons that a human player would normally press to perform an action in the game space. These are shown in the top right portion of the game window. The neural network itself is visually represented by the large collection of coloured lines and boxes at the top of the window. The program uses the NEAT algorithm in order to evolve the neural net, which is a modified GA that I will discuss in the next section. Whilst the MarI/O agent is running, the displayed network dynamically updates the colours of its connections in order to show which connections are currently active e.g if Mario is moving to the left, the connections that lead to the output node marked 'Left' would be active. Other than that, the rest of the interface is identical to that of an unmodified copy of Super Mario World.

What I like about the solution:

While the input of a simplified view of the game is unorthodox in this context, it appears to be very effective, as in the Youtube video, an extensively evolved MarI/O employs a very effective method of traversing the level involving very precise timing in order to cover lots of ground quickly and avoid enemies. In the source code*, this input is calculated succinctly by simply analysing the game window in blocks and passing that to the NN.

The visual representation of the network is rather intriguing. I believe it would be very effective in my similar solution, as Aman Doshi stated in [Meeting 4](#). Moreover, the colour coding of connections and highlighting of currently active ones is also something we discussed. Viewing how the topology of the network changes over generations may help the user understand what's going on in the code itself.

What I would do differently:

The UI attempts to provide the user with the clearest representation possible of the algorithms and structures at play, but the size of the space required to display the input window as well as the network itself makes the whole game view seem crammed together and claustrophobic, and may overload the user with too much information at once.

In addition, this program was only trained extensively on this particular level of Super Mario World. As a result, it performs extremely well on this level, but in a subsequent video that SethBling published*, we see the program trying to beat a handful of other levels. In all cases, it either wasted time getting past a certain segment of the level due to its different layout or did not complete the level at all because it could not evolve sufficiently to get past certain obstacles in the allotted time. Thus, I would compile a large dataset of levels, perhaps a large portion of all the levels in the whole of Super Mario World and have MarI/O run on each one sequentially. This may prevent the issue of MarI/O overfitting* to the level 'Donut Plains 1'.

Additional Learning Algorithm: NEAT

The MarI/O neural network ([Additional Existing Solution: MarI/O](#)) evolves using a genetic algorithm called NEAT (Neuroevolution of Augmenting Topologies). This is a solely neuroevolutionary genetic algorithm, meaning it is an algorithm that only works when the genomes being evolved are neural networks. What separates NEAT from other genetic algorithms is that it evolves the weights of the neural networks as well as the structure of the network itself. That is to say, it starts with a bare bones NN, with just the input and output nodes, and no hidden layers in between. As it evolves, it generates new 'species', which are sets of genomes which have a different topology of their neural nets. All of these alterations are stored in some kind of record structure.

This algorithm was first devised by Kenneth O. Stanley and Risto Miikkulainen, two Computer Science professors at the University of Texas at Austin. They detailed the algorithm in their seminal research paper in 2002* which marked a breakthrough in neuroevolution.

NEAT has its relative benefits and disadvantages, which I need to evaluate to determine whether it is worth it to conduct further research on it and implement it as my evolution

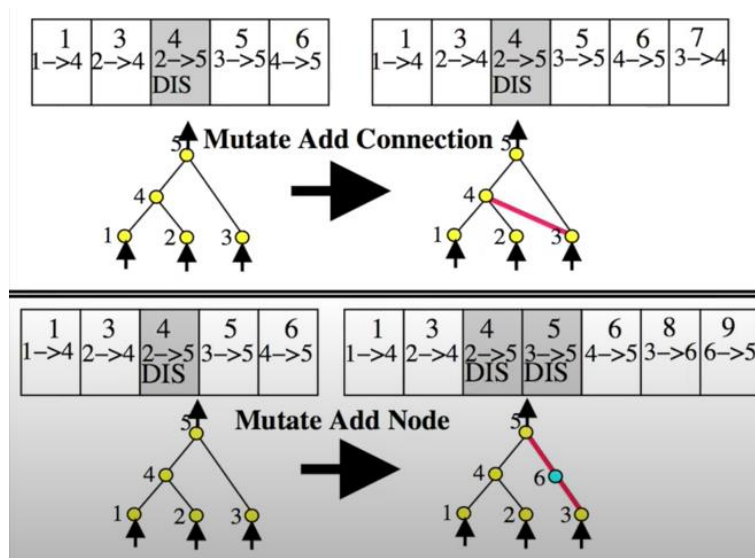


Figure 10: Simplified view of how NEAT works

algorithm rather than the normal, fixed topology genetic algorithm that I am already familiar with and is easier to implement.

Benefits:

In some instances, NEAT has been shown to reach near optimal solutions faster than other GAs. However, the difference in time taken is not particularly of importance to me, because as long as my ER agent can eventually learn to run as well as a human (within reason), my investigation is successful. Moreover, I will likely include an option for the user to speed up time in case they want to make the evolution run faster, so again, the speed benefits are not a significant factor in the context of my project.

NEAT uses many of the same principles as a base GA does, as after all they both stem from the process of mimicking real evolution. Thus, I would not have an unreasonable amount to learn in order to get a grasp of the algorithm.

The 30-page research paper by the UT Austin professors has lots of information regarding the structure, functionality and genome encodings of a NEAT algorithm, which I can refer to should I get stuck.

Disadvantages:

The biggest disadvantage of NEAT as a whole is the fact that it loses out on parallelisation. This means that it becomes increasingly difficult to have two neural nets evolving using NEAT simultaneously, as their topologies are bound to mutate differently from each other simply by chance. This quickly becomes convoluted to handle programatically, and in the specific case of my investigation, more difficult to display visually. This goes against my desire to have multiple agents running simultaneously to allow the user to better visualise a population of creatures, similar to behaviour in nature.

There is also the fact that I am unfamiliar with NEAT compared the algorithm I planned to use. This may increase the likelihood of running into bugs that I am at first unsure how to solve, as well as potentially messy code as a result of code of not knowing to structure a NEAT algorithm to a suitable standard for this project.

Evaluation of NEAT

While the NEAT algorithm is certainly a viable approach for a solution to my investigation, I will be sticking with my original idea of a base GA with a fixed NN topology. I believe the reasons for this have been outlined well under the benefits and disadvantages headers of the previous section. The biggest factor dissuading me to use the NEAT algorithm is the lack of parallelisation capabilities, as one aspect of the investigation that I will likely make a mandatory requirement is the displaying of a population of agents running simultaneously.

Consideration of Neural Network Visualisation

After pondering on Aman's suggestion of having a visualisation of at least one NN in my finished system, I have decided to not consider this as a potential requirement. The main reason for this is, as Aman mentioned above in our meeting, the massive strain this would put on the CPU; in a proposed solution that has this as a requirement, I can presume that there will need to be a few steps for visualising the NN that happen **at least every frame**:

1. Find best performing agent
2. Access that agent's NN
3. Update the visualisation with the corresponding visual representation e.g., update color of weights according to whether they are positive or negative, or update thickness of weight based on magnitude.

The process of each of the agent's NNs feeding forward each timestep already creates a cumulative strain on the CPU, so this additional component would likely render the game virtually unusable, even with an optimised library. Moreover, the extent to which a dynamically updating NN can be rendered faithfully on to a screen is also a questionable matter.

Neural Network Structure and Functionality

Now that I have made a definitive choice on the set of structures and algorithms I will use, I can produce an abstract idea of how I will encode these into my ER game. Most notably, I can decide on the hyperparameters of the neural network that each generation will possess. This includes the number of nodes per layer as well as the number of hidden layers. The number of inputs and outputs I choose will directly determine the number of nodes in their respective layers e.g 5 inputs = 5 nodes in the input layer.

NOTE: Visual representation of the completed neural network structure can be found in the Modelling section

Input Layer:

The input nodes represent the information that will be provided to each agent, and thus determines the extent to which the agents can learn. As such, it is important to encode these inputs in a format that is useful to the network but also succinct to the point where unnecessary information is filtered out.

Taking into account the Existing Solutions and how they went about encoding inputs, I have devised a method that will work in my specific context. I plan to provide each agent with 5 'sensors' surrounding their player character. These will be invisible beams protruding from different angles around the character. These beams can detect objects that they collide with as well as information about said collisions, including distance from

the source of the sensor to the object as well as the type of the object itself. This seems like suitable information to pass into a network as inputs. Thus, the input layer for my NN will have **10 input nodes**, two for each sensor, with an extra node for the bias node. One of each sensor's nodes will store the distance from the agent to the object the sensor detected, and the other node will store the numerical tag of the object it detected. I can define these numerical tags beforehand, perhaps using a dictionary.

Determining Number of Hidden Layers:

The hidden nodes are responsible for enabling the network to make learn more complex patterns in whatever they are applied to. In a more mathematical sense, they allow the network to model more complex non-linear functions. There is no rule or standard for determining the actual number of hidden layers as well as number of nodes in those layers, but there are a few general guidelines to follow to ensure that the network can perform as well as possible. PhD Jeff Heaton presents this table in a post on his research

Table: Determining the Number of Hidden Layers

Num Hidden Layers	Result
none	Only capable of representing linear separable functions or decisions.
1	Can approximate any function that contains a continuous mapping from one finite space to another.
2	Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy.
>2	Additional layers can learn complex representations (sort of automatic feature engineering) for layer layers.

Figure 11: Jeff Heaton's table for determining hidden layer count

blog*:

The blog post only discusses network architectures as they related to supervised and unsupervised learning problems, not reinforcement learning problems. However, I do suspect that the number of hidden layers would have a similar effect to what is shown in the table. Therefore, more than 2 hidden layers would mean that the network can learn complex relationships from the inputs. I believe the environment that my neural net will be running in is fairly complex, as the computer must learn to react to changes in the game environment with certain rules and nuances in order to play as a human does. Thus, according to Heaton's table, I believe using **3 hidden layers** will be sufficient for this investigation.

That being said, there are various blog posts that mention the notion of hyperparameter optimisation, which is running a separate algorithm to determine the optimum layer and node count for the network. This appears to be a fairly complex process, and the speed with which the NNs achieve near optimal performance are not of great concern for this project, so I will refrain from using this technique.

Determining number of hidden nodes in each hidden layer:

Heaton also writes about general rules of thumb regarding the number of hidden nodes to include in each layer. They are as follows:

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be $\frac{2}{3}$ the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

These rules are such that they balance the costs and benefits of having too many or too little hidden nodes. Too many hidden nodes would result in overfitting, as well as run the risk of making the neural network run inordinately slow. Contrarily, too little neurons would result in underfitting*. Thus, it is imperative to find a reasonable equilibrium.

Initially, I will follow the second rule that Heaton mentions, so the number of hidden nodes per hidden layer will be: $10 \times \frac{2}{3} = 3$ (rounded down) + 5 = 8 hidden nodes, including the biases.

Output Layer:

Ultimately, the agents will need to take actions based on the calculations performed by the network, and these actions will be moves that a human player is permitted to perform. Thus, each output node of the network can be mapped to a specific button press, each of which corresponds to an action in the game. There are 5 of these actions that the agents can do. Listed below are these actions along with the key that a human player would have to press to carry out this action:

- Move left, A key
- Move right, D key
- Do nothing, no keys pressed
- Jump, Spacebar
- Slide, S key

The output layer's activation function will likely be one that normalises the values in each node to a number between 0 and 1. An example of such a function would be the sigmoid

function. This is to ensure that the output values are not skewed. The network can then choose the output node with the highest activation and perform the action that corresponds with that particular output node. This functionality will be described further in the Design section.

Final Requirements

Taking a hollistic review of all my meetings and research, I can devise the final requirements for my solution in order to maximise its efficacy as a somewhat interactive simulation/ visualisation, taking into account the features that are of top priority and keeping the scope discussed with Mr. Forsyth ([Meeting 2](#)) in mind.

NOTE: The requirements are divided into two main sections: User Requirements and Algorithmic Requirements. They are visually separated in this list but still sequentially numbered.

User Requirements:

These requirements are what the user must be able to see in the completed solution:

1. Upon startup of the finished build of the game, the user must see a start menu, where they must click a Start button in order to continue.
2. After starting the game, the user must be prompted to enter two parameters that will be passed as parameters into the genetic algorithm, these being:
 - a. The total population size of every generation.
 - b. The mutation rate of the evolution.
3. The parameters entered in Requirement 2 must be validated to ensure that:
 - a. The population size is an integer greater than 2 and less than 100, inclusive.
 - b. The mutation rate is a float between 0 and 1.
 - c. An appropriate error message should be displayed in the case of an invalid input.
4. After validation of parameters in Requirement 3, the user must confirm their choices, upon which the visualisation starts. The user must be able to see:
 - a. The main game window, which includes:
 - i. The current generation of AI agents running along the course (initialised to generation 0 – randomly generated population).
 - ii. The course, complete with all features that were available in the base game e.g spikes, sliding obstacles, enemies (***NOTE: This has all been already implemented in the base game and so will not need to be re-implemented***).
 - b. The various utilities and statistics about the current state of the simulation:
 - i. The maximum fitness score, which is the maximum fitness that any one agent has gotten over the course of the whole evolution.

- ii. A graph plot of average fitness over time, with average fitness on the y-axis and generation number on the x-axis.
 - iii. The current generation number of the evolution.
 - iv. The number of agents left running at that point in time for that generation.
 - v. The mutation rate
 - vi. The population size
- c. Buttons with text that delineates their purpose:
 - i. A 'Reset Evolution' button that discards all current generations and begins again with a new population starting from generation 0.
 - ii. An 'End Evolution' button that quits out of the visualisation and redirects the user back to the Start Menu.
- d. A slider that controls the timescale of the visualisation (speeds it up or slows it down)
- e. If an agent hits a spike, floating boombox or enemy or veers to the edge of the course on either side, they will die and they will be eliminated from the run, leaving one less agent in the current run:
 - i. Once all agents of a generation have been eliminated, the next generation begins with a new population generated by the genetic algorithm (This process will be detailed further in the ***Algorithmic Requirements***)

Algorithmic Requirements:

The inner workings of the visualisation must function as such:

- 5. A custom Neural Network class must be defined:
 - a. The neural net class must define fields in its constructor:
 - i. A vector of size 10 representing the input layer
 - ii. Each hidden layer gets its own matrix of weights
 - b. There must be a FeedForward method that feeds forward the network using matrix multiplication, returning the outputs in a 5-dimensional vector.
 - c. The sigmoid activation function must be defined.
 - d. The hyperbolic tangent function must be defined.
- 6. An AI agent must be represented as a Neural Network object attached to an existing Player object which has a PlayerMovement script that was present in the base game as well as a 3D model
 - a. The agent must have an InitialiseNN method that instantiates the neural network for that particular agent.
 - b. The agents must continually feed forward their NNs to generate new actions to take using the NNs FeedForward method.
 - c. The agents must each have a CalculateFitness method
 - i. The method takes in the current game score of that agent and returns a fitness score for that agent based on that distance.
 - ii. A higher game score means a higher fitness score.

7. Once all agents in a generation have been eliminated, the processes of the genetic algorithm are carried to generate the new population. The steps are as follows:
 - a. The top two agents (agents with the 2 highest fitness scores) are preserved and added to the new population.
 - b. Two parent agents are selected from the old population. These can be one, both or neither of the two agents that were preserved.
 - i. Agents with higher fitness scores are more likely to be selected as parents
 - ii. An agent can be paired with itself for the parent selection
 - c. The neural nets attached to the parent agents are crossed over using a crossover function*
 - i. The weight values of the parents' neural network are crossed over to generate a new child weight matrix.
 - ii. A new neural net instance is created with the child weight matrix
 - iii. Using the mutation rate as a probability, mutate this child neural network by altering the weight values.
 - iv. The generated neural net is attached to a newly instantiated agent.
 - v. This agent is added to the new population.
 - d. All steps of Requirements 7b through 7c must be iterated over to generate a specific number of child agents such that the new population size is the same as the previous population size.
 - e. The new population of agents is rendered in the game view, and the new generation begins
 - i. The program's generation counter is incremented
8. The processes listed in Requirement 7 must be repeated indefinitely until the user chooses to end or reset the evolution.

Scope

The table below will outline the scope of my project, indicating what will be done and what will not. However, the exact details of the project will remain in the final requirements.

<u>In scope</u>	<u>Out of scope</u>
Multiple agents running in parallel	NEAT algorithm
Graph plot of fitness over time	Multiple fitness metrics*
Utilities	Advanced crossover algorithms
Base genetic algorithm	Probability distribution for mutation rate e.g Gaussian
Two-point crossover	Player verses AI functionality*

User input of parameters	Dynamic changing of parameters during evolution*
Feed forward, densely connected neural network	Hyperparameter optimisation
	More complex network topologies e.g Recurrent Neural Network

Evaluation of Scope

I will now explain my decisions to not include the elements in the table marked with a *. The out-of-scope elements that are not marked as such have either been mentioned previously or are, in my opinion, self-explanatory in their reasoning for not being implemented (would take up too much screen space, too advanced, would render the simulation too slowly, wouldn't drastically enhance the efficacy of the solution etc.).

- **Multiple fitness metrics:** I stated in previous sections that it would be interesting to incorporate different metrics for calculating fitness and use these different fitness in separate evolutions in order to generate agents which exhibit varied behaviours. I believe that in an ER, there is not much room to generate a wide variety of behaviours amongst agents, so it would not be worth the time to implement this feature. Moreover, this may confuse the user watching as they may not fully grasp what the fitness function is actually doing, so introducing multiple different fitness functions would only worsen their experience.
- **Player vs AI:** This was the initial investigation I detailed in my [Overview](#), but has since changed due to evaluation of ability to be rigorously tested and measurable. This is explained further in the [Consideration of Predicted Requirements](#) section.
- **Dynamic changing of parameters:** Giving the user the flexibility of changing the population size and mutation rate whilst an evolution is running would throw the whole evolution into disarray, as there is a possibility that the user drastically alters the mutation rate, which could potentially cause a near optimal agent to be mutated and worsened in terms of fitness, making the evolution lose progress and potentially never gain it back.

Modelling

This section will be for the various models I produce for different key aspects of the problem.

Abstract model of neural network:

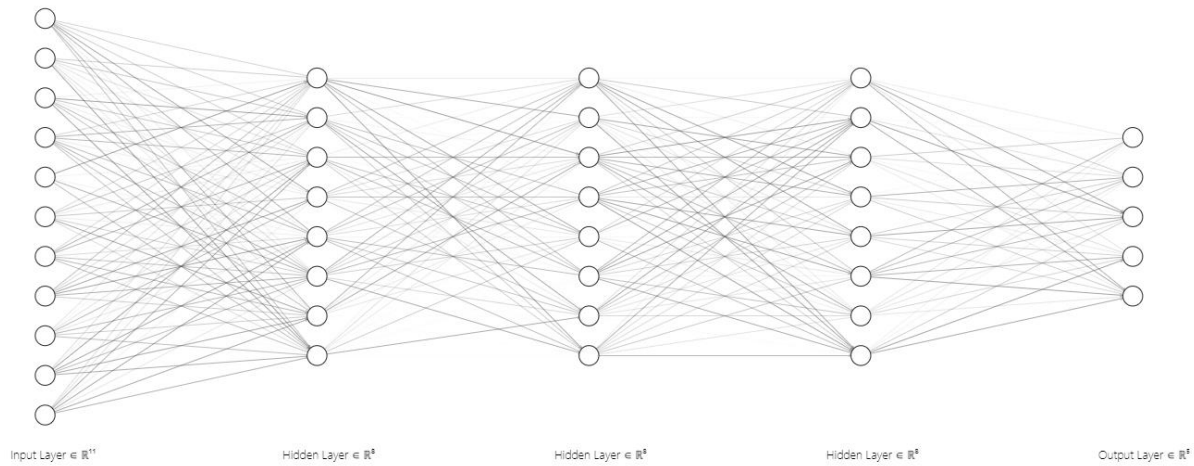


Figure 12: Model of agent NN as an undirected, dense graph

Hand-drawn neural network and further details:

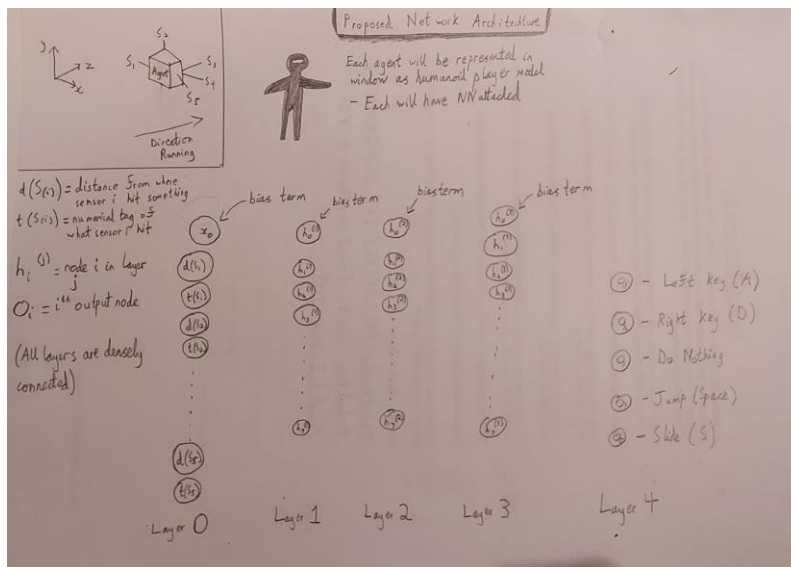


Figure 13: Model of agent NN as an undirected, dense graph

Figure 13 does not include the connections between layers for the sake of neatness and the diagram not being overly convoluted. In the top left there is an attempt at depicting the sensor system that will be used to pass the 10 inputs into the neural net. The additional bias units are placed on the top row of the NN and labelled with an arrow. The sensor system is briefly detailed below. It will be more clearly visualised during modelling and prototyping in the [Design](#) section.

Sensor System:

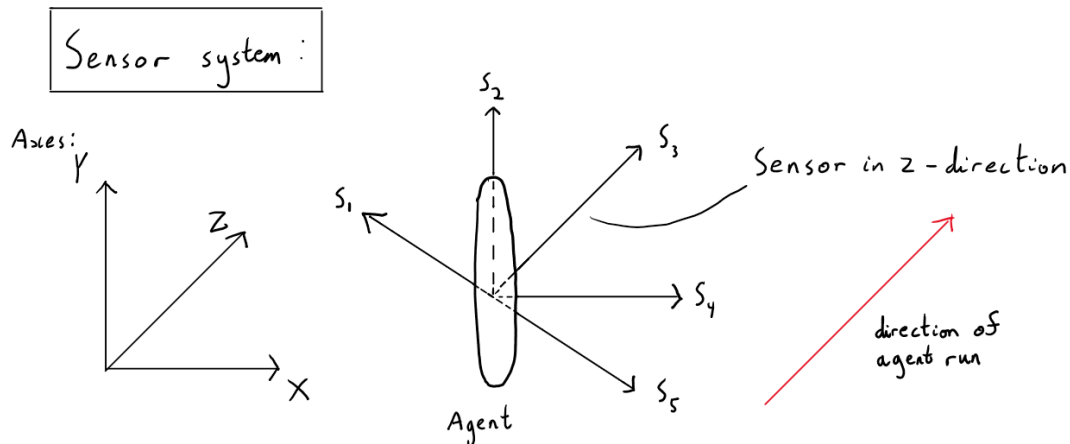


Figure 14: Visual Representation of Sensor System

Here, the sensors are represented as vectors, but they are conceptualised as infinite rays that are cast out from the centre of the agent's model. The capsule like shape representing the agent is for the sake of representation: their models will actually be humanoid 3D models.

Pseudocode:

I will now display my initial pseudocode for a neural network class. This is simply the NN that will be attached to each agent; no aspect of the genetic algorithm is included.

```
// Neural Network data structure represented using a class
// Class is modular so any set of hyperparameters can be used

/* Functions: Vector(n) returns an empty n-dimensional vector; Matrix(n, m) returns an
empty n x m matrix */

CLASS NeuralNet

    CONSTRUCTOR(numInputs, numHiddenNodes, numHiddenLayers, numOutputs, numLayers)

        this.inputs <- Vector(numInputs)

        this.outputs <- Vector(numOutputs)
```

```
this.hiddenLayers <- [ ]  
this.weights <- [ ]  
FOR i <- 0 TO numHiddenLayers  
    this.hiddenLayers.add(Vector(numHiddenNodes))  
ENDFOR  
ENDCONSTRUCTOR  
SUBROUTINE Initialise()  
    // Populate weight array with matrices  
    FOR i <- 0 TO numHiddenLayers  
        IF i = 0  
            // This is for the first set of weights, from the input layer to the  
            // first hidden layer  
            this.weights.add(Matrix(numHiddenNodes, numInputs + 1))  
        ELSE IF i = numLayers - 1  
            // Last hidden layer to output layer weights  
            this.weights.add(Matrix(numOutputs, numHiddenNodes + 1))  
        ELSE  
            // Hidden layer to hidden layer weights  
            this.weights.add(Matrix(numHiddenNodes, numHiddenNodes + 1))  
        ENDFIF  
    /* This method is not included in this section of pseudocode. All it does is  
    randomise all the weight values in each matrix in the weight array */  
    RandomiseNeuralNet()  
ENDSUBROUTINE  
SUBROUTINE FeedForward()  
    this.inputs <- Tanh(this.inputs)  
    // Matrix multiplication to get the activation of each node  
    this.hiddenLayers[0] <- Tanh(this.weights[0] * this.inputs)  
    FOR i <- 1 TO numHiddenLayers  
        this.hiddenLayers[i] <- Tanh(this.weights[i] * this.hiddenLayers[i-1])  
    ENDFOR  
    this.outputs = this.weights[numLayers-1] * this.hiddenLayers[numHiddenLayers-1]
```

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College.
Centre Number: 74615. Component Code: 7517/C.

```
// Output normalised between 0 and 1

RETURN Sigmoid(this.outputs)

ENDSUBROUTINE

// Sigmoid and Tanh can take either numerical values or a matrix

SUBROUTINE Sigmoid(z)

// e = Euler's Number (2.7818...)

RETURN 1 / (1 + e-z)

ENDSUBROUTINE

SUBROUTINE Tanh(z)

RETURN ez - e-z / ez + e-z

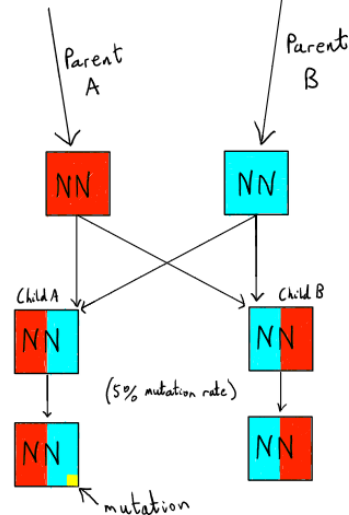
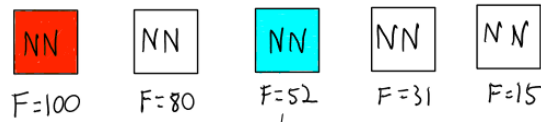
ENDSUBROUTINE
```

High Level Model of GA:

This graphic shows a simplified model of how neuroevolution works in the general case, not just in the specific case. It is useful for me to be able to understand these individual steps to the point where I can hand draw them, as I will be working with a GA for the majority of my Technical Solution.

→ Create initial population

NN × 5 (5 neural nets)



Loop until
desirable
fitness
reached

Figure 15: High Level model of Fixed Topology Genetic Algorithm

The values chosen for the fitness scores, population size and mutation rate are arbitrary and solely for demonstration.

Analysis Conclusion

I have collated research from numerous different notable sources regarding the nature of my solution, the techniques I will use as well as the requirements that my solution will fulfill. The meetings I conducted all contributed to bettering my understanding of what my investigation entails, as well as how to make it valid by introducing measurable, quantitative testing criteria. The existing solutions I looked at let me evaluate what my solution should and should not include based on the merits and shortcomings of the existing solutions. The method I carried out to measure sample users' performances in my game was ensured to be fair and valid, and so is an essential factor for determining the success of my solution. The modelling I have done will play a pivotal role in providing a basis for my Design section, which will involve definitively producing the structure of my solution, most notably the neural net pseudocode; it can quite simply be translated into

any language of my choosing – in this case, C# (all technologies used for this investigation are discussed under the [Technologies](#) heading of the [Design](#) section.

Design

In this section I will document the key aspects and structure of my investigation. I will do this via modelling which expands upon models seen in my [Analysis](#) as well as prototyping and testing the core elements and refining them if need be.

High Level Overview

These are the elements that will be present in the final simulation. They are derived from my [Requirements](#) but are more focused on how they will be implemented in a real coded solution.

Start Menu:

Recalling Requirement 1:

- The Start Menu must display the title of the project, which will be “Generation” followed by the number of the first generation that passes [Success Criteria 1](#) e.g. “Generation 57” means that at least one agent from generation 57 of the first ever evolution was able to survive for longer than 28 seconds.
- The title screen must give the player the option to:
 - Start the visualisation by pressing a “Start” button

Parameter Input:

Recalling Requirements 2 and 3:

- When the user is prompted for input, a dialog box must appear with two fields for the user to fill out if they wish:
 - Mutation rate, defaulting to 0.05
 - Population size, defaulting to 50
 - The visualisation must not begin until the user has entered valid values for each field. This means a number between 0 and 1 (inclusive) for Mutation rate, and a positive integer greater than 2 and less than 200 for population size.
 - A Confirm button must be present to allow the user to continue.

On-Screen Statistics:

Recalling Requirement 4b, excluding 4bii:

- Whilst the agents are running, several parameters will be shown in the top portion of the screen so as to not crowd the game window where the visualisation is happening. These statistics are:
 - Current generation number
 - Max fitness of any agent in the evolution
 - Number of remaining agents
 - Mutation rate
 - Population size

NOTE: The above bullet points that are highlighted in red indicate the statistics that need to dynamically update over the course of the evolution.

Neural Network:

Recalling Requirement 5:

- The NN will have the structure defined in the [Algorithmic Requirements](#). That is, 1 input layer of size 11, 3 hidden layers, each of size 8, and an output layer of size 5.
- These will all be represented as vectors and matrices, using the Linear Algebra module of the Math.NET Numerics* library (detailed in [Technologies](#)).
- In the Neural Network script, there will be methods for initialising the weights as matrices, randomising said weights, feeding forward the NN using matrix multiplication and the two activations I will be using (Sigmoid and Tanh).

Agent:

Recalling Requirement 6:

- An agent that runs in the visualisation will be comprised of four main components:
 1. A neural network script which initialises and manages that agent's NN
 2. A movement script which processes the NN's outputs and returns the action that the user will take for that timestep.
 3. A method of handling inputs for the NN (see Sensor System model in Analysis)
 4. A player model, which is a free 3D model* from the Unity Asset Store*
- The agents will be fully animated, with animations for running forwards, jumping, sliding, falling and dying.
- The performance of an agent will be gauged by the fitness score of that agent. This is discussed in the following heading.

Population of Agents:

Recalling Requirement 4a ii, iii, iv:

- Upon starting the visualisation, a population of agents will be generated according to the population size parameter that the user specified.
 - These agents must be identical in model and dimensions and must all have the same scripts attached to them.
 - These agents must not collide with each other whilst running.
 - No agents should be generated more than once (two agents may be initialised with the same weight values giving the illusion that they are duplicates, but I mean this in terms of the program actually generating the exact same agent twice).
 - One agent dying should not affect the outcomes of any other agents.
 - A new population of offspring is to be generated at the start of each generation.

Genetic Algorithm:

Recalling Requirement 7:

- The generated population must be passed into an evolutionary algorithm after all agents from each batch or 'generation' have been eliminated. When this happens, the following processes should take place.
 - The NNs of the population should be sorted in descending order of fitness score. This will likely be done using an array.
 - The two first NNs from this sorted array should be preserved and added to another array that will represent the next generation. This concept is called **elitism**.
 - In the current population array, two NNs will be randomly selected to be parents. This random selection is weighted by the fitness of each NN, so higher fitness NNs are more likely to get picked.
 - The selected parents will be crossed over by having their weight values being combined via the two-point crossover function. This will generate two new child NNs.
 - The children will be passed into a mutation rate function. With the probability that was recorded in the Parameter Input section of the Start Menu, each weight value will be altered to a random value.
 - The children should then be added to the next generation
 - The selection, crossover and mutation processes should repeat for this generation until the new generation array has the same size as the previous generation array.
 - The new generation NNs should all be attached to individual Agent objects and instantiated for the next generation to begin.

Graph plot:

Recalling Requirement 4bii:

- The graph that plots average fitness against number of generations must be present alongside the other metrics listed above.
- The graph will dynamically update its display with each generation; the data point for each new generation will be added when the next generation begins.
- The graph will be a line chart, which allows the user to easily see the general trend in the data.

Buttons and sliders:

Recalling Requirements 4c and d:

- The following buttons must be displayed in the game window:
 - 'Reset Evolution': A red button that will destroy all current agents and begin a new evolution starting at generation 0.
 - 'End Evolution': A red button that will destroy all current agents and redirect the user back to the start menu.
- A slider labelled 'Timescale' should be present alongside the button. When scrubbed, it should alter the speed at which the evolution is progressing by modifying Unity's internal timescale.

Audio:

- Whilst the agents are running, original music will be playing in the background. It is a short synth loop that I composed whilst I was creating the prototype ER in summer 2020.
- There are custom sound effects for jumping, bouncing on bounce pads and dying. These were made using SFXR*, a browser-based sound effects generator.

Taking all these components into account, I am conceptualising the project at a high level in 3 larger sections, with the finer details such as sound and 3D models excluded. They are:

1. **The neural network:** The actual NN data structure, including the Sensor System for inputs, the actual topology of the NN, the matrix operations needed to feed forward, and the two activation functions needed.
2. **The genetic algorithm:** The GA will take many NNs in a given population and perform its processes on it, generating a new population for the next generation of active agents. These processes can be further subdivided into population initialisation, fitness calculation, parent selection, crossover and mutation.
3. **The UI:** The on-screen details about what's happening in the visualisation, including the graph, buttons, sliders and statistics.

There are also the agents themselves, which the NNs will ultimately control, but these only require linking to the NNs, and so are not the elements of the program that largely comprise the AI section.

These components all need to communicate with each other in one way or another. This is roughly outlined in the figure below.

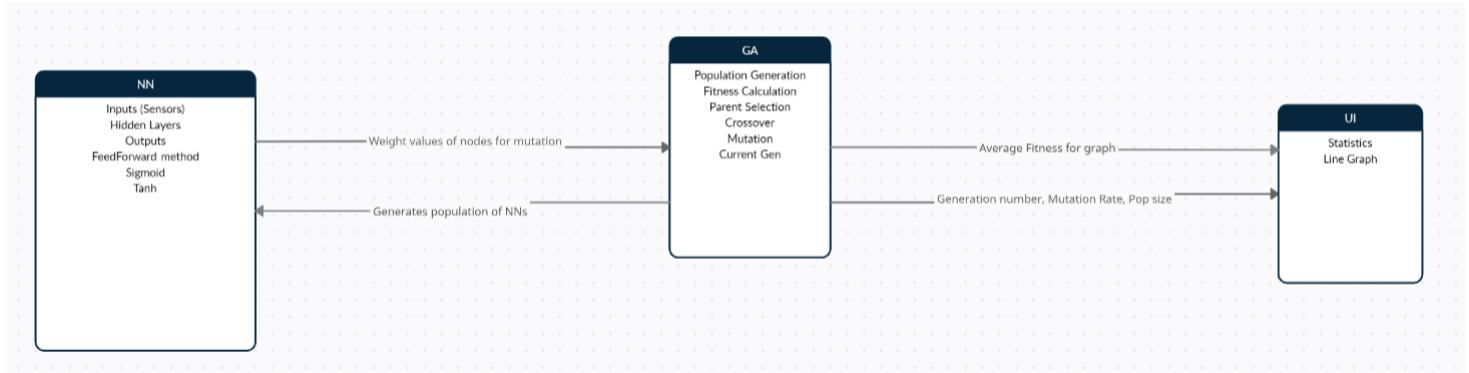


Figure 16: Data Flow Diagram of 3 larger subsections

Technologies

Since I first devised main elements of this project in my initial [Analysis Overview](#), I have been more or less set on the choice of Unity as a game engine for this project, with C# as my language of choice. This section will evaluate this decision as well as highlight the additional third-party modules used for this project.

Unity and C#:

I am familiar with Unity as I used it to develop the prototype for my initial ER game in summer 2020. The visual editing tools it offers as well as its seamless communication with Visual Studio Code, a popular IDE, are all main reasons why I think it is suitable with this project; I will need to switch back and forth between my IDE and the Scene View within Unity, so having an automatic compile and execute at every single save saves large amounts of time. Moreover, Unity provides a structured, non-ambiguous interface with the objects present within the game, which can either be edited via the visual editor or through script. A mixture of both of these is required to simulate the complex behaviour that I am striving to achieve.

C# has a few advantages over JavaScript, which is the other language compatible with Unity, with regards to this project. Some of these are:

- It is an object-oriented language, and so supports the concepts of encapsulation, inheritance* and polymorphism*.
- It is statically typed, and so is explicit in its type casts.
- Automatic garbage collection*.
- More concise syntax.

- Generally faster to run (C# is a compiled language whereas JS is interpreted).
- Supports the integer and array data types.

Most of these are simple quality of life improvements that will just prove for a more streamlined experience with C#, whilst properties like its OOP capabilities are suited better to this project specifically - using C# over JS in this project will undoubtedly prove to greatly enhance its efficacy for all the reasons mentioned above, along with the simple fact that I am more experienced with C# for game development, specifically with Unity.

C# Libraries:

The two main features that need to be adapted from third party sources are the linear algebra structures needed as well as facilities for drawing charts, which Unity does not have built in. For handling matrices and vectors, I have chosen the MathNET.Numerics library, which can easily build any dimensions of matrices and perform operations on them, all whilst also being massively optimised.

The ability to render charts in the game window will also need to be provided by a third-party package. I have found two that provide all the functionalities I desire from the graph. These functionalities are:

- Line Graph Format
- Gridlines
- Chart Title
- Axis Labels
- Ability to alter scale

The two tools I have found for this are: A GitHub repository* by user codemaker2015 which acts as a Unity package, and an asset called 'Dynamic Line Chart' from the Asset Store. I believe that the latter will be of better use for this project as, after some brief trials, I can see that it has improved functionality over the first example.

Overview of User Experience

This section will, through the use of models and descriptions, provide a high-level overview of what the user should be able to see and do in the finished system.

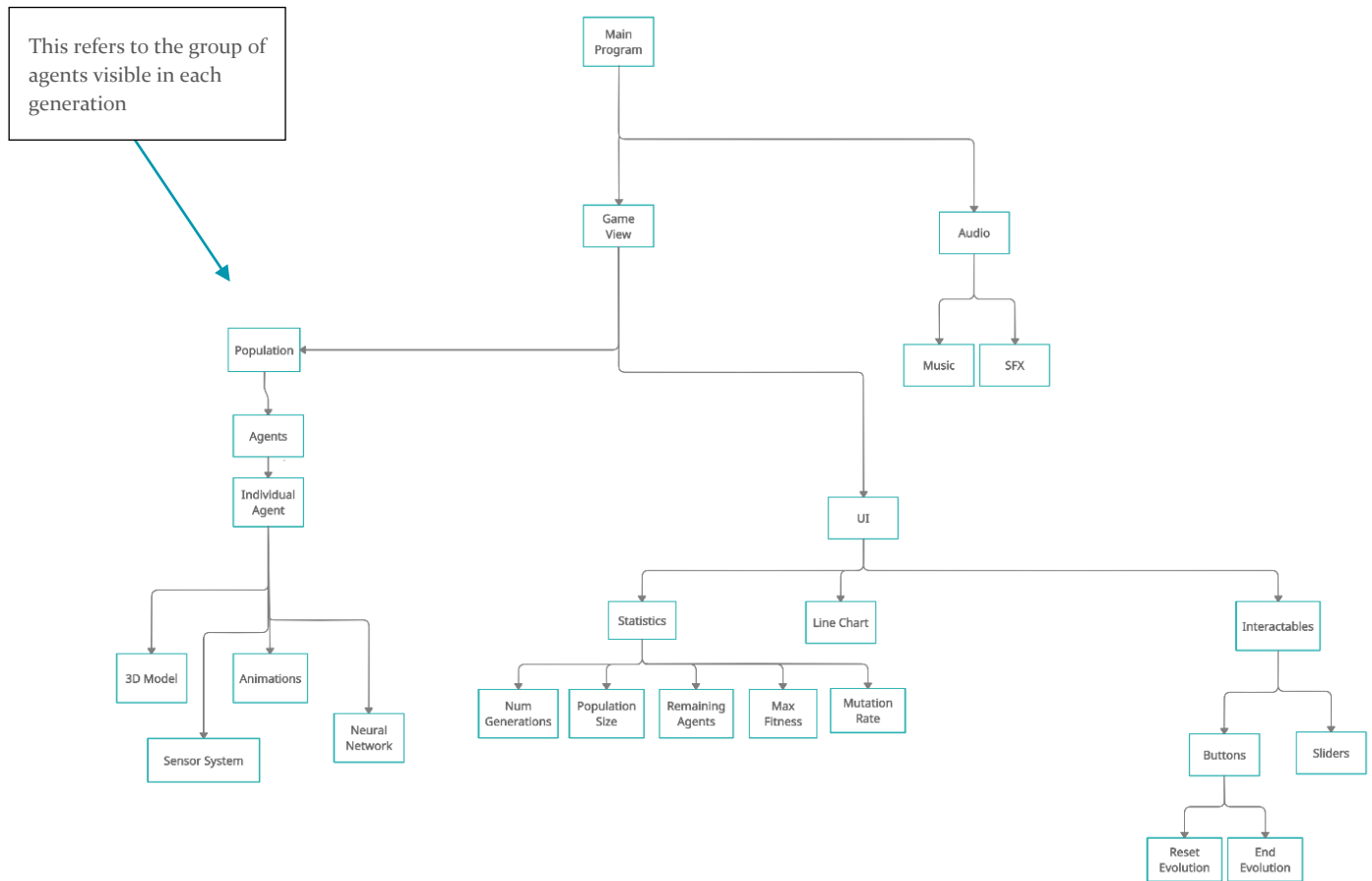


Figure 17: Hierarchy Diagram of User Experience

Figure 17 portrays the complete high-level structure of the user's experience upon starting the visualisation. As such, it does not show the inner workings of how the program achieved this (this is instead shown in the following sections). The specific details of elements that were present in my existing prototype ER (e.g., the course structure, obstacles, bounce pads) have been omitted, as they already exist, and so implementing them is not a focus in this project. Regardless, they will still be present in the finished program. An important thing to note from this diagram is that the components that the user sees that are directly controlled by AI techniques (everything under the Population) are separated from the UI elements as they are in two separate branches from the main Game View. This will most likely reflect the actual implementation process, as it is better to deal with these two larger components in separate phases rather than implement them in parallel. Moreover, these two layers need to communicate with each other, so having one completed before the other will likely prove useful. As a whole, this diagram more clearly communicates the decomposition of the problem that I have arrived at as opposed to describing it in prose.

Figure 18 is a simple flowchart detailing how the main processes of beginning the visualisation via the Start Menu as well as the user inputting desired parameters of the GA. The two subroutine cells are for validation subroutines for the mutation rate and population size respectively. They will check whether the parameters of the correct data type – if this is true, they will be checked to ensure they are in the allowed range of values (ranges for both parameters are defined in the [Requirements](#)).

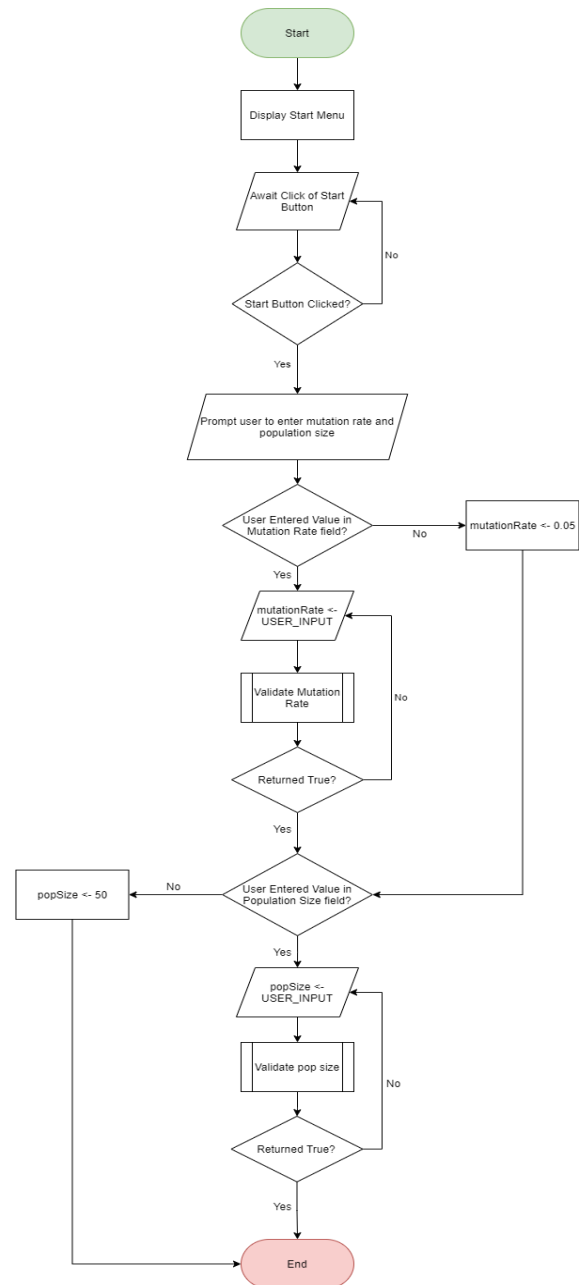


Figure 18: Flowchart of Start Menu and User-Defined Parameter Input

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College.
Centre Number: 74615. Component Code: 7517/C.

Pseudocode for overall functionality for main Game Loop:

// mutationRate and popSize will be assigned via user input before the main visualisation

// SETUP (happens at runtime)

SUBROUTINE Start

numGenerations <- 0

maxFitness <- 0

agentsRemaining <- popSize

Render empty Line Chart

Render buttons and slider

InitialisePopulation()

Render Statistics

ENDSUBROUTINE

// What happens in the Game Loop (this runs EVERY FRAME)

SUBROUTINE Update

// Button Presses

IF EndEvolution Button Pressed

EndEvolution()

ELSE IF ResetEvolution Button Pressed

ResetEvolution()

ENDIF

// Slider Values

IF TimeScaleSlider.ValueChanged()

Unity.TimeScale = TimeScaleSlider.Value

ENDIF

// Handle all eliminated agents

deaths <- Number of agent deaths

Destroy all 'dead' agent objects

agentsRemaining <- agentsRemaining - deaths

IF agentsRemaining <- 1

IF numGenerations <- 0

```
        maxFitness <- FitnessScore(last remaining agent)
    ELSE
        currentFitness <- FitnessScore(last remaining agent)
        IF maxFitness < currentFitness
            maxFitness <- currentFitness
        ENDIF
    ENDIF
ENDIF
IF agentsRemaining <- 0
    Update statistics // This includes incrementing generation num and updating maxFitness
    Render new statistics
    Calculate average of final fitnesses across all agents in generation
    Plot datapoint on Line Chart for this generation
ENDIF
ENDSUBROUTINE
```

The pseudocode above details the processes that should occur once the actual visualisation has begun. It is high-level, and many of the individual lines can be decomposed into their own scripts, each with individual setup and game loop subroutines (Unity is very conducive to this). It serves to highlight the main processes of the visualisation, in other words, **what the player actually sees happening**. Some of the larger components of this will be decomposed and detailed in following sections.

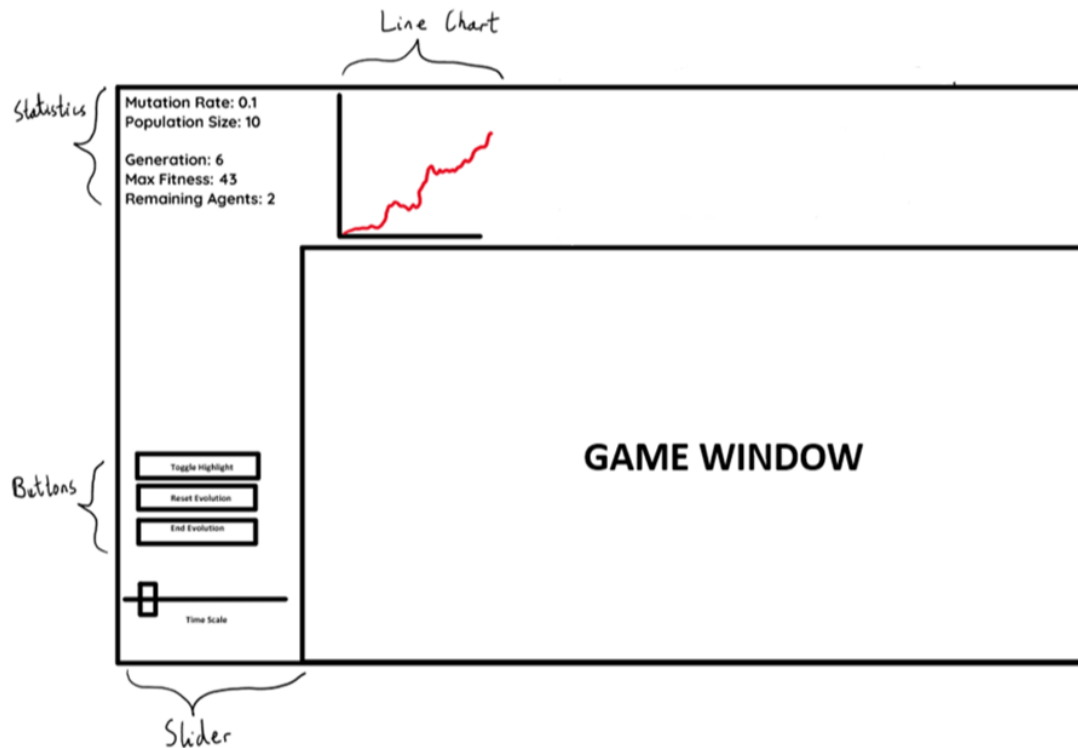


Figure 19: UI Wireframe of Visualisation

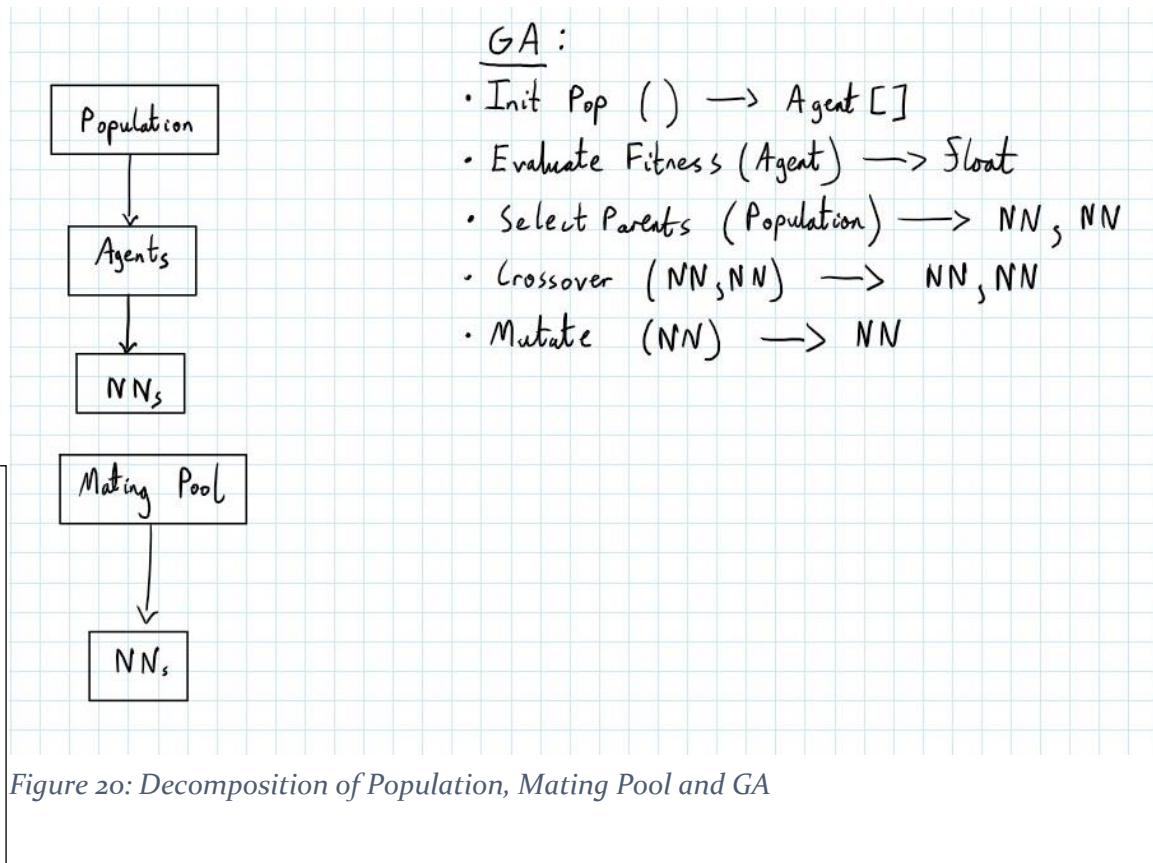
The figure above is a rough outline of how I imagine the interface in the finished program to look like, with the core components from Figure 17 portrayed. The statistics and graph values were chosen arbitrarily for the sake of depiction. Wireframes like this are generally used for the UI of web and mobile apps, but I believed it would be useful to do one for this project as there is a lot of information that needs to be displayed on the screen, with some needing to be dynamically updated from frame to frame, or even more often than that in the case of physics events.

The main rationale for the placement of the elements present is that I want loosely related functionalities to be spatially coupled together e.g the statistics and line chart are depicting traits of the evolution itself, so they occupy the top band of the screen. Another motivation for this mockup is reducing clutter on the screen, as the user may get overwhelmed if the various UI elements are chaotically arranged, and thus may not get as much out of the program as initially desired.

It is important to emphasise that this mockup is very rough, and as such the exact scalings and alignments of the elements are not perfectly accurate to those of the finished solution.

This mockup does not reflect the final UI; the finished system may look different to this due to unforeseen consequences of using the mockup UI. One of these consequences may be that the camera for the game view simply is unable to effectively capture all the agents

running with this UI design. This can probably be fixed by adjusting the camera coordinates and rotation. The only way to know this for sure is via prototyping.



It is important to note that an **Agent is composed of an NN, but an NN does not have to have an attached Agent**. This saves processing time, as an NN doesn't take up as much space as an Agent object. Because of this property, I can handle the parent selection process by directly working with NNs rather than working with agents and having to constantly access their attached NNs. After the appropriate processing of the NNs has been done, I can initialise the new generation by instantiating an agent for each NN and attaching them.

Identifying Core Elements

I will identify core components of this program so that I can begin designing and prototyping them. These are the parts of the project that **need** to be present in the finished system in order for it to operate at a minimum level. As a result, all of the components apart from key UI features and essential technicalities of my machine learning model and algorithm will not be present.

The core elements that I have identified are:

- **Start Menu** – This will be how the user actually begins the main portion of the program, and so it needs to be there to allow them to start on demand.
- **Neural Network** – My NN class will be the engine that drives each agent to actually exhibit the desired behaviour in the finished system, so I need to test this out to verify that I can represent an abstract model such as an NN as a C# object with the Sensor System I devised, as well as whether these outputs can be properly sent between scripts in order for the GA to actually handle these values. The NN section will be split into the subsections:
 - a. **Sensor System and Inputs**
 - b. **Network Structure**
 - c. **Feed Forward and Activation Functions**
- **Agent** – The agent and the NN will go hand-in-hand as the network will be attached to a particular agent throughout its run. The NN will be the component that influences the agent's behaviour e.g when it should go left, when it should jump, when it shouldn't do anything etc. I need to test whether this actually occurs once I have finished the NN prototype.
- **Population of agents** – Once an individual agent prototype is shown to be working as expected, I must observe how multiple agents running simultaneously behave, and whether there need to be certain changes to the existing scripts of the base game in order to accommodate for there being essentially multiple players running on the course.
- **Genetic Algorithm** – The algorithm will be the element that governs all NNs and agents of the project; it is how the NNs within the agents learn, and thus without it, the whole premise of the project would not be there. It will be best if I decompose the algorithm into separate modules and prototype, test and refine them separately, whilst keeping in mind that they will all have to communicate with each other. That being said, it is important to note that **only 3 sections of the GA need to be present for it to work at a minimum level**. These are the **population initialisation, fitness function and mutation**. Hence, the remaining ones, which are to do with agent reproduction, are not considered core components to this project.

I am not considering the statistics or line graph as core elements as they are dependent on the majority of the elements listed above- the visualisation can still run without these elements as they are solely UI-based.

Menus

Description:

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College.
Centre Number: 74615. Component Code: 7517/C.

The Main Menu is one of the simpler elements of this project but is crucial in the sense that it is the way in which the user interfaces with the program and actually starts it; it is simply a means for the user to click a button and then begin the more prominent components of the program. That being said, I have decided to add a component in the Start Menu that is not listed in my Requirements: An Options button, that will allow the user to adjust the general settings of the program. Currently, the only two adjustable settings will be music volume and SFX volume.

Pseudocode:

```
SUBROUTINE MainMenu()  
    Render Title  
    Render StartButton  
    Render OptionsButton  
    WHILE True  
        IF StartButton.Clicked()  
            ParameterInput() // Method will be implemented in later sections  
        ELSE IF OptionsButton.Clicked()  
            OptionsMenu()  
        ELSE IF QuitButton.Clicked()  
            Quit()  
        ENDIF  
    ENDWHILE  
ENDSUBROUTINE  
  
SUBROUTINE OptionsMenu()  
    Render Title  
    Render BackButton  
    Render MusicVolumeText  
    Render MusicVolumeSlider  
    Render SFXVolumeText  
    Render SFXVolumeSlider  
    IF BackButton.Clicked()  
        StartMenu()  
    ENDIF
```

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College.
Centre Number: 74615. Component Code: 7517/C.

```
IF MusicVolumeSlider.ValueChanged()  
    Game Music Volume <- MusicVolumeSlider.Value  
IF SFXVolumeSlider.ValueChanged()  
    Game SFX Volume <- SFXVolumeSlider.Value
```

Models:

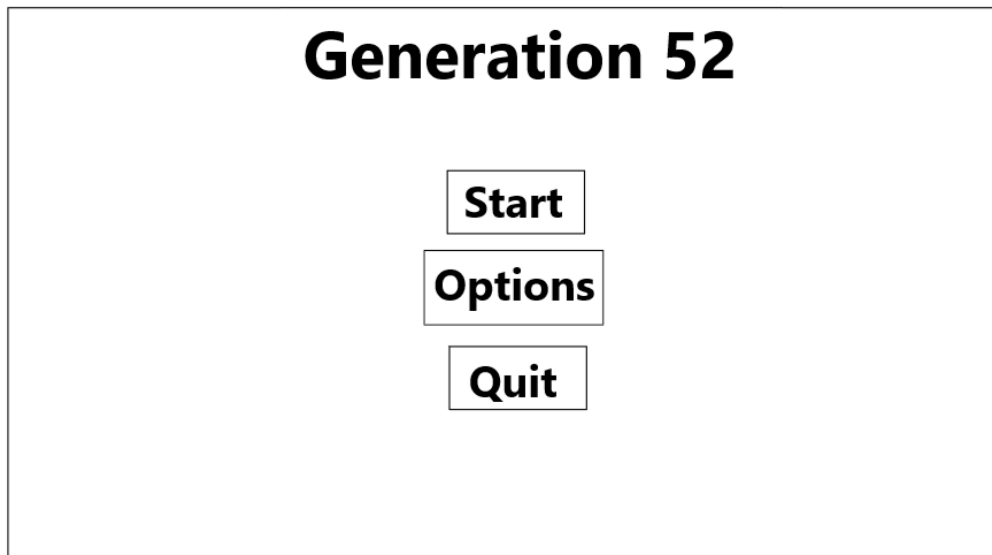


Figure 21: Wireframe of Main Menu

The number 52 was chosen arbitrarily and for the sake of demonstration only.

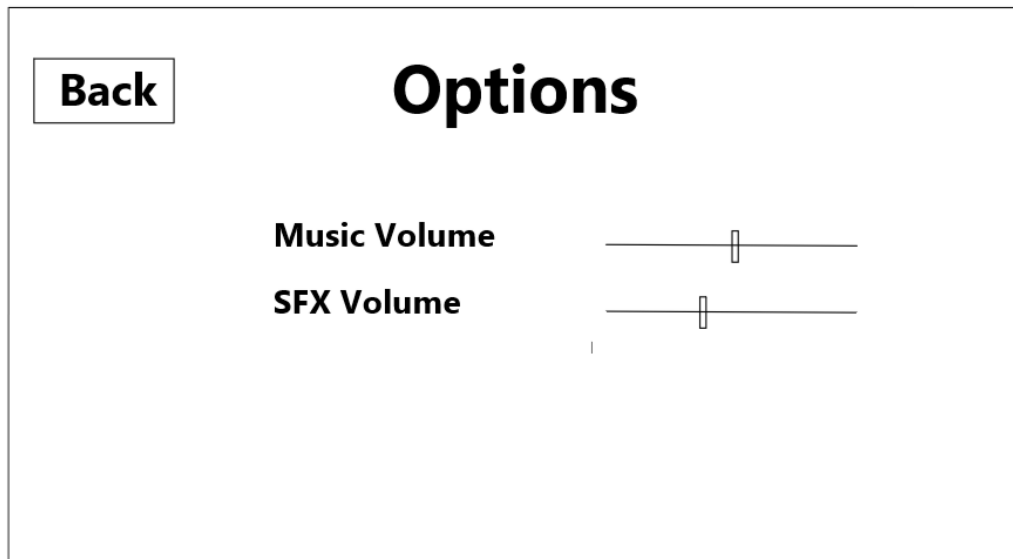


Figure 22: Wireframe of Options Menu

Unity Prototype:

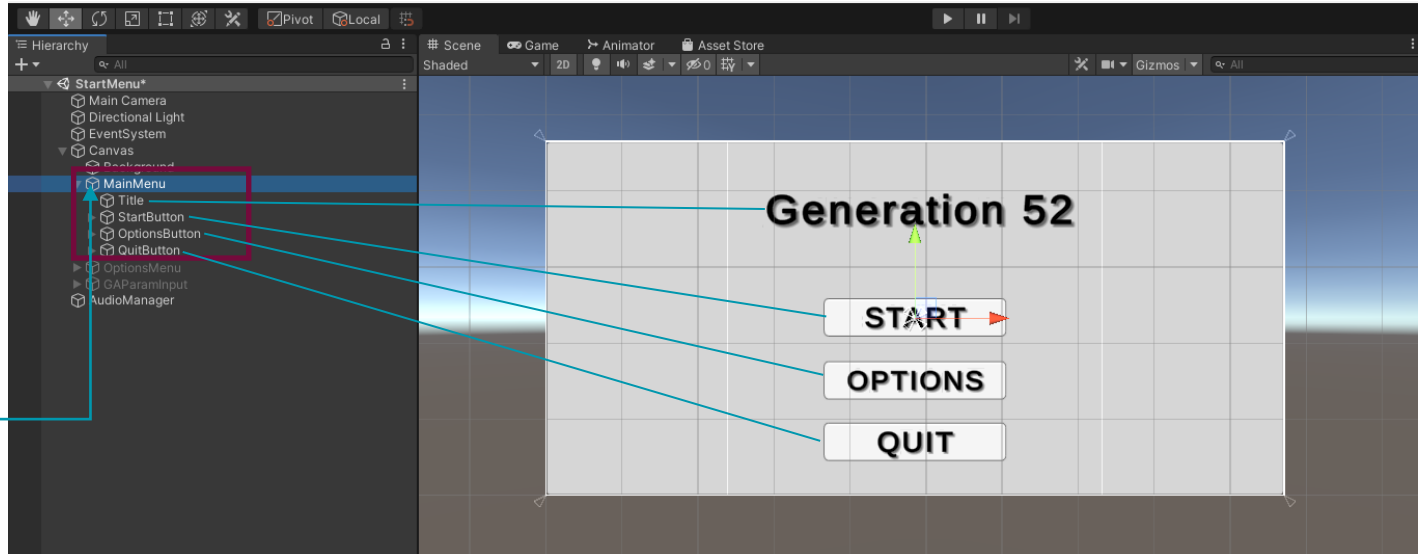


Figure 23: Editor View of Prototype Start Menu

MainMenu Object and its Child Objects

OptionsMenu Object and its Child Objects

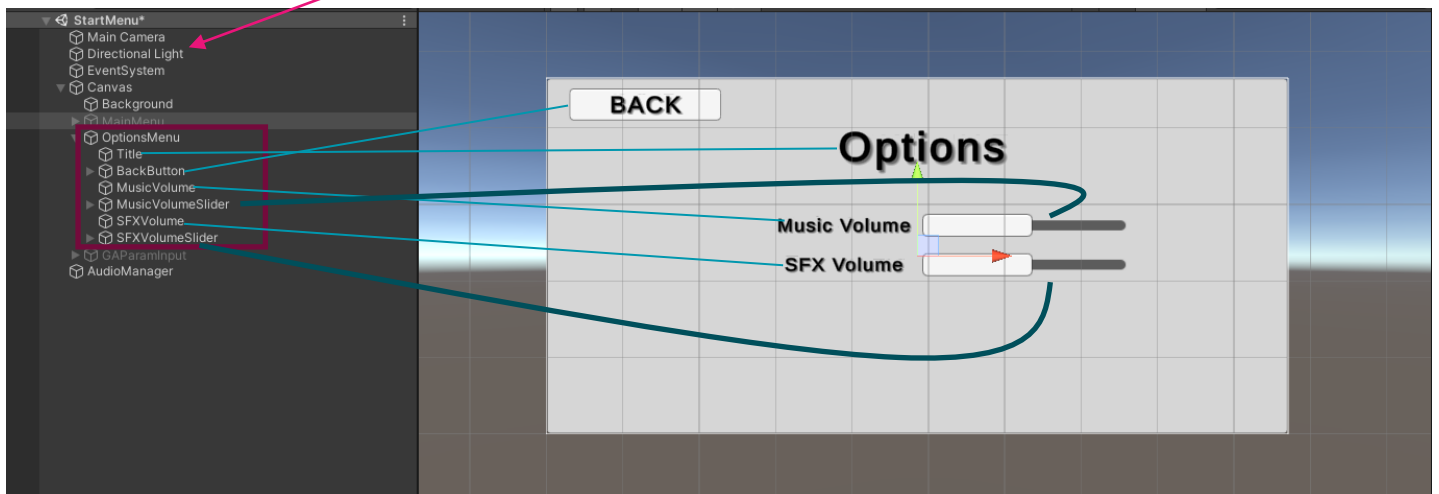


Figure 24: Editor View of Prototype Options Menu

```
public class MainMenu : MonoBehaviour
{
    public GameObject options;
    public GameObject paramInput;


    public void Start()
    {
        gameObject.SetActive(true);
        options.SetActive(false);
        paramInput.SetActive(false);
    }

    // Executed when Start Button clicked
    public void StartVisualisation()
    {
        // Loads next Scene in the queue (this is the main visualisation scene as there are only two scenes)
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

    // Executed when Quit Button Clicked
    public void Quit()
    {
        // Application only quits in an actual build of the project, NOT in the Unity Editor, so this debug log is to verify the function executes on quit button click
        Debug.Log("QUIT");
        Application.Quit();
    }
}

public void UpdateVolume(Sound s, float volume)
{
    // Change Audio Manager volume value for this sound
    s.volume = volume;

    // Update actual sound volume from the sound source in the game
    s.source.volume = volume;
}
```



In Start() Method, the Main Menu is ensured to be the only one out of the eventual 3 menu screens to be active

I added the code shown in the UpdateVolume() method as part of an AudioManager script which is attached to a custom C# object, also called AudioManager, that was

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College.
Centre Number: 74615. Component Code: 7517/C.

present in the prototype ER. This UpdateVolume() method takes a Sound object s (which is also a custom C# Object) and updates its volume and source attributes, which in turn changes the volume of the in-game sound denoted by the parameter s.

```
public class OptionsMenu : MonoBehaviour
{
    public Slider musicSlider;
    public Slider sfxSlider;
    public AudioManager am;
    void Start()
    {
        am.sounds[0].volume = musicSlider.value;
    }

    // Executed on change of music slider
    public void UpdateMusicVolume()
    {
        // Set value of music volume to music slider value
        am.UpdateVolume(am.sounds[0], musicSlider.value);
    }

    // Executed on change of SFX volume
    public void UpdateSFXVolume()
    {
        // All other sounds other than 0th item are sfx, so update volume of all of them to sfx slider value
        for (int i = 1; i < am.sounds.Length; i++)
        {
            am.UpdateVolume(am.sounds[i], sfxSlider.value);
        }
    }
};
```

In Figures 23 and 24, blue lines are drawn between the object in Unity's Hierarchy view and the visual representation of that object in the Scene view.

The code snippet above is the code attached to the OptionsMenu game object. I can probably refine it by not hard coding the 0th element of the sounds array in the Start() method, and instead making the index of the main theme a public variable that can be changed in the Unity Editor. This is shown below.

```
public class OptionsMenu : MonoBehaviour
{
    public Slider musicSlider;
    public Slider sfxSlider;
```

```
public AudioManager am;
private Sound themeMusic;
void Start()
{
    themeMusic = Array.Find(am.sounds, sound => sound.name == am.musicName);
    themeMusic.volume = musicSlider.value;
}

// Executed on change of music slider
public void UpdateMusicVolume()
{
    // Set value of music volume to music slider value
    am.UpdateVolume(themeMusic, musicSlider.value);
}

// Executed on change of SFX volume
public void UpdateSFXVolume()
{
    // All other sounds other than 0th item are sfx, so update volume of all of them to sfx slider value
    for (int i = 1; i < am.sounds.Length; i++)
    {
        am.UpdateVolume(am.sounds[i], sfxSlider.value);
    }
}
}
```

This will store a reference to the theme music's Sound object

Getting the reference via a lambda expression instead of hard coding the index of the theme in the sounds array

The public variables at the top of the script hold references to the slider that controls music volume, the slider that controls SFX volume and the AudioManager object, in that order.

The prototype switches between the Start Menu and Options Menu through the use of Unity Event Triggers. This allows individual Game Objects to call methods attached to them when a certain event is enacted. Consider the below example:

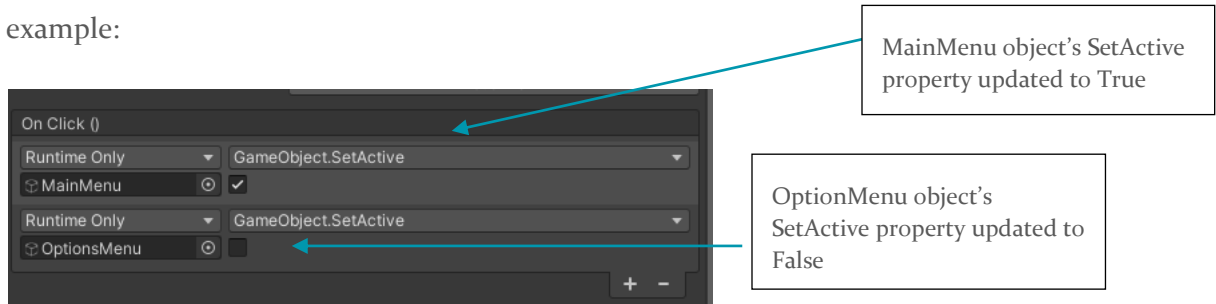


Figure 25: OptionsMenu BackButton Event Triggers

Figure 25 shows the Event Triggers that are given to the Back Button in the Options Menu. When the BackButton is clicked, the MainMenu object is set to active, which displays all the Main Menu content. It also disables the currently active OptionsMenu object so that its contents are not displayed, hence there is no overlapping content.

Prototype Tests:

I am rather unfamiliar with the way that Unity's Event System works, but it seems rather intuitive. I will test these prototypes to make sure that all the desired features of this prototype are present.

Test 1: Ensuring Start Button takes user to ParamInput menu.

- **Expected Test Output:** On click of the Start Button, the user should be taken to the menu where they can enter their custom GA parameters, which for now should just be a blank grey canvas (this menu will be detailed in the Implementation phase).
- **Testing Output:**

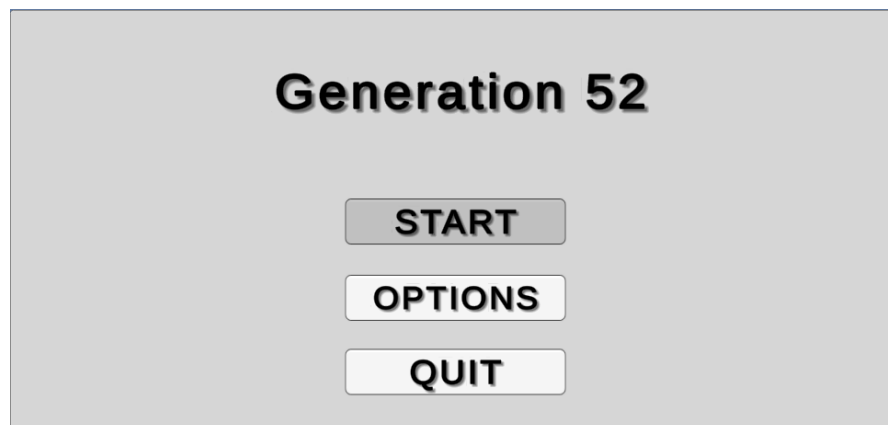


Figure 26: Test of Start Button Click



Figure 27: Newly rendered screen after start button click

Test Outcome: The test worked as expected – the blank screen was rendered on click of the Start Button

Test 2: Ensuring Options Button takes user to Options Menu

- **Expected Test Output:** On click of the Start Button, the user should be taken to the Options Menu.
- **Testing Output:**



Figure 28: Test of Options Button click

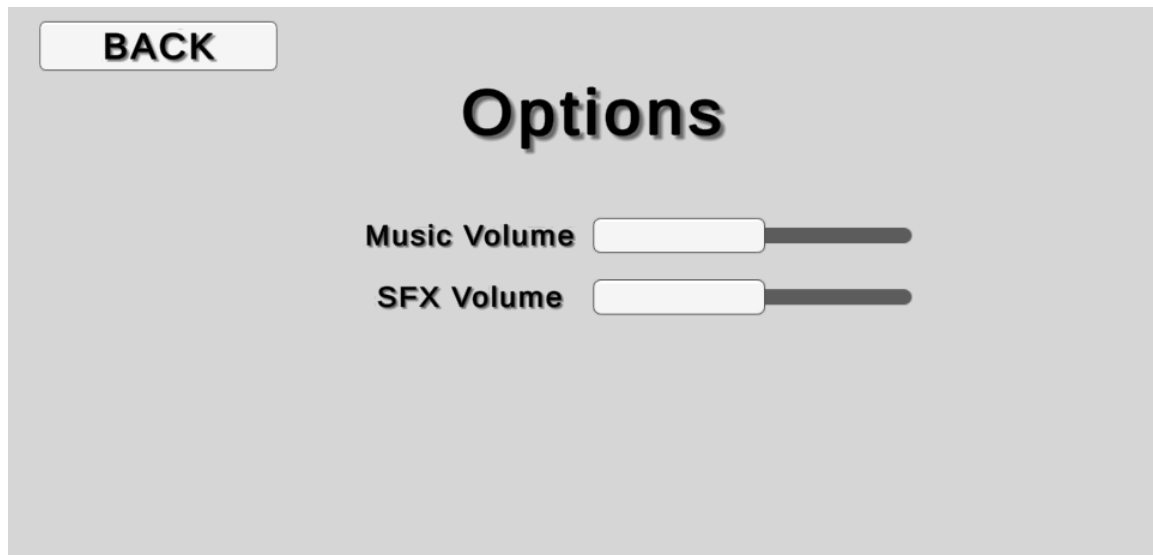


Figure 29: Resulting screen from Options Button Click

- **Test Outcome:** This test worked as expected: the Options Menu was correctly displayed on click of the Options Button.

Test 3: Ensuring Quit Button can force a quit

- **Overview:** The Quit Button will, of course, quit the program and close it. This can only be done in a built version of a Unity project, **not** in the Unity Editor. To test that the functionality of my Quit Button will work whilst still keeping the project within the Editor, I will simply send a message to the Debug Console on click of the Quit Button.
- **Expected Test Output:** On click of the Quit Button, the message “QUIT” should be logged in Unity’s Debug Console.

- **Testing Output:**

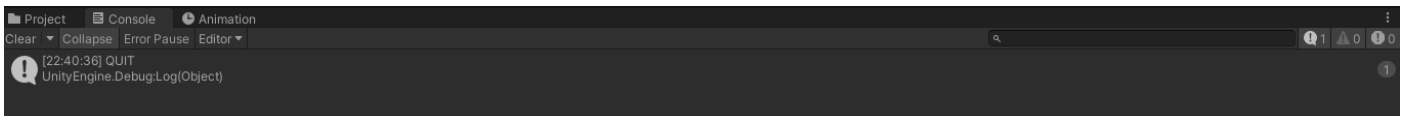


Figure 30: Test of Quit Button click

- **Test Outcome:** The test worked as expected – I can verify that the click of the Quit button will quit the application in the finished build.

Sensor System

Description:



The NN of each running agent must have a set of inputs at every timestep that will then determine the optimal action to take (according to that NN). The inputs will be passed into the NN using the Sensor System, which is briefly detailed in my Analysis [Modelling](#) section.

The Sensor System will be a custom object attached to each agent. Each Sensor System has a script of the same name attached to it that governs the creation and handling of each sensor.

Recalling the structure of my NN, the summary of components of the Sensor System script is as such:

1. Five direction vectors: left, forward-left, forward, forward-right and right.
2. Five Rays around the front half of the agent model all being sent out from the center of the agent. These rays are created according to the five direction vectors.

3. These Rays won't be visible in the final build of the visualisation, but in the Unity Editor they can be shown using a debugging method which can draw lines according to a vector. This will help with implementation of later components.
4. Any time a ray collides with something, the distance from the agent of that collision will be recorded, as well as with what object that collision was. This will be done using Unity's RayCast data type.

Pseudocode:

SUBROUTINE SensorSystem(sensorLength)

dirVectors <- [left, forwardLeft, forward, forwardRight, right]

FOR vector v IN dirVectors

Emit Ray from center of agent in direction of v

Draw green line of current Ray with length sensorLength

IF Ray Collided with object

hitDistance <- Distance of agent from object

colObject <- Numerical representation of object that was collided

with

Pass hitDistance and colObject into NN input nodes

Models:

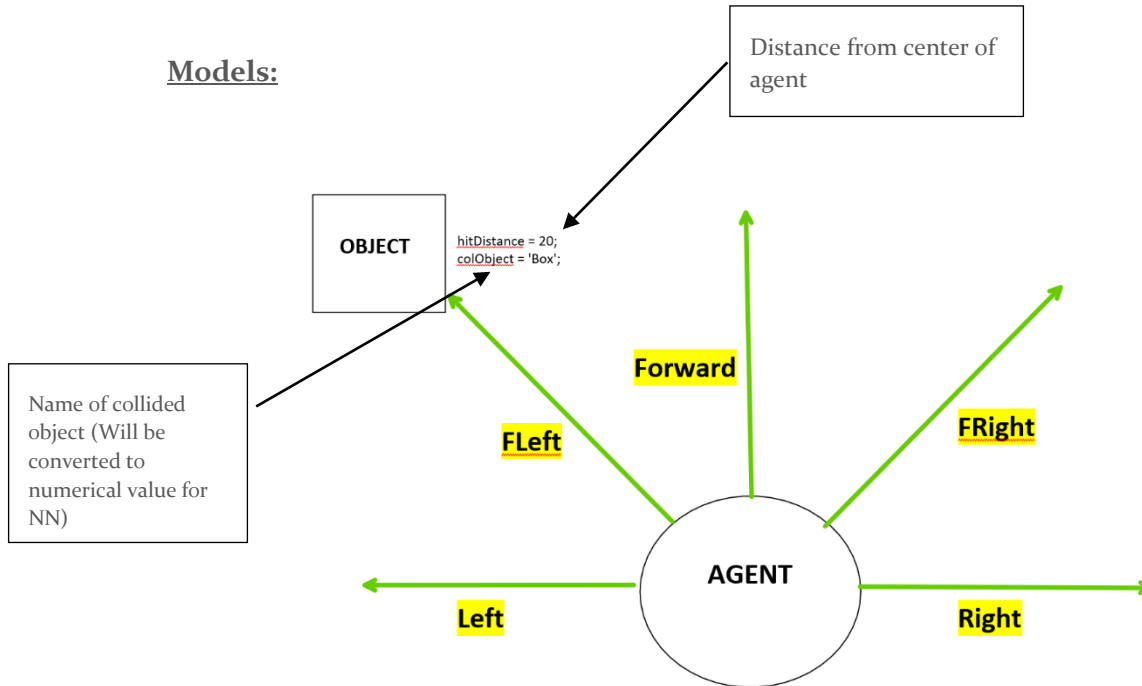


Figure 31: Diagram of single agent Sensor System with sample collision

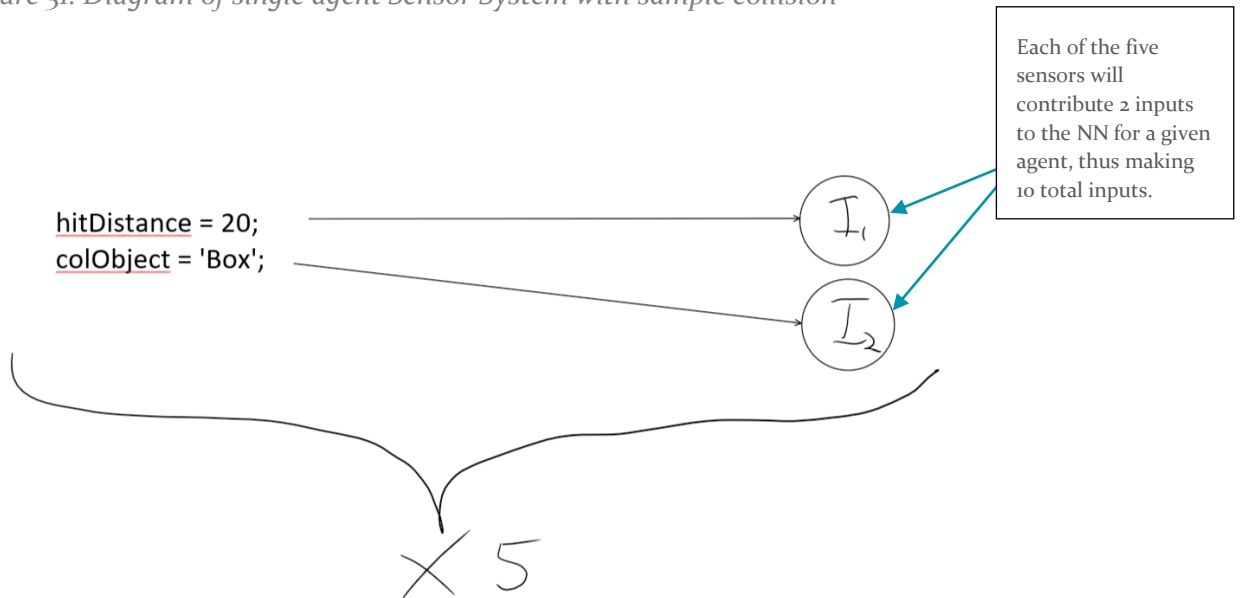


Figure 32: Diagram of data flow between Sensor System outputs and NN inputs

The Sensor System's ultimate purpose is to pass its outputs into the NN where they will act as inputs for that sensor. There will be 5 input pairs in total for each of the 5 sensors (for one agent). The processes involved in that will be detailed in the NN Design section.

Unity Prototype:



Figure 33: SensorSystem Object attached to Agent in Editor View

```
public class SensorSystem : MonoBehaviour
{
    public float sensorLength;

    public GameObject[] obstacles;
    Dictionary<string, int> tagsToNum = new Dictionary<string, int>();

    public AgentNN agentNN;

    int cols;

    void Start()
    {
        int val = 0;
        foreach (GameObject obstacle in obstacles){
            tagsToNum[obstacle.tag] = val;
            val++;
        }

        cols = agentNN.nn.inputLayer.ColumnCount;
    }

    void FixedUpdate()
    {
        HandleSensors();
    }
}
```

List of references to GameObjects that I have labelled as obstacles

Dictionary to map the tags of each obstacle (string) to a numerical value

In the Start method, we are populating the tags dictionary with unique integer values.

Sensors are updated every Physics event (which is independent from framerate – it happens as many times as needed.)

```
Vector3[] InstantiateDirVectors()
{
    // Five vectors at increasing angles of 45 degrees clockwise
    Vector3 left = -transform.right;
    Vector3 fLeft = -transform.right + transform.forward;
    Vector3 forward = transform.forward;
    Vector3 fRight = transform.forward + transform.right;
}
```

```
Vector3 right = transform.right;
Vector3[] dirVectors = {left, fLeft, forward, fRight, right};
return dirVectors;
}

void HandleSensors()
{

    Vector3[] dirVectors = InstantiateDirVectors();

    // Arrays to pass into NN
    List<float> hitDistances = new List<float>();
    List<int> collidedObjects = new List<int>();

    // Initialise sensors by casting Rays out and check for collision
    Ray ray = new Ray(transform.position, dirVectors[0]);
    RaycastHit hit;
    foreach(Vector3 v in dirVectors)
    {
        ray.direction = v;

        if (Physics.Raycast(ray, out hit))
        {
            Debug.DrawLine(ray.origin, hit.point, Color.red);
            if (hit.collider.gameObject.tag != "Agent" && hit.collider.gameObject.tag != "Unta
gged")
            {
                hitDistances.Add(hit.distance / norm);
                collidedObjects.Add(tagsToNum[hit.collider.gameObject.tag]);
                Debug.DrawLine(ray.origin, hit.point, Color.green);
            }
            else
            {
                // If sensors did not detect an obstacle, add dummy values to list as we need
the order of the sensors to be preserved when passed into the input layer
                hitDistances.Add(-1f);
                collidedObjects.Add(-1);
            }
        }
        else
        {
            // Again, add dummy values if there was no RayCast
            hitDistances.Add(-1f);
            collidedObjects.Add(-1);
        }
    }
}
```

```
}  
  
}  
LinkToInputLayer(hitDistances, collidedObjects);  
}
```

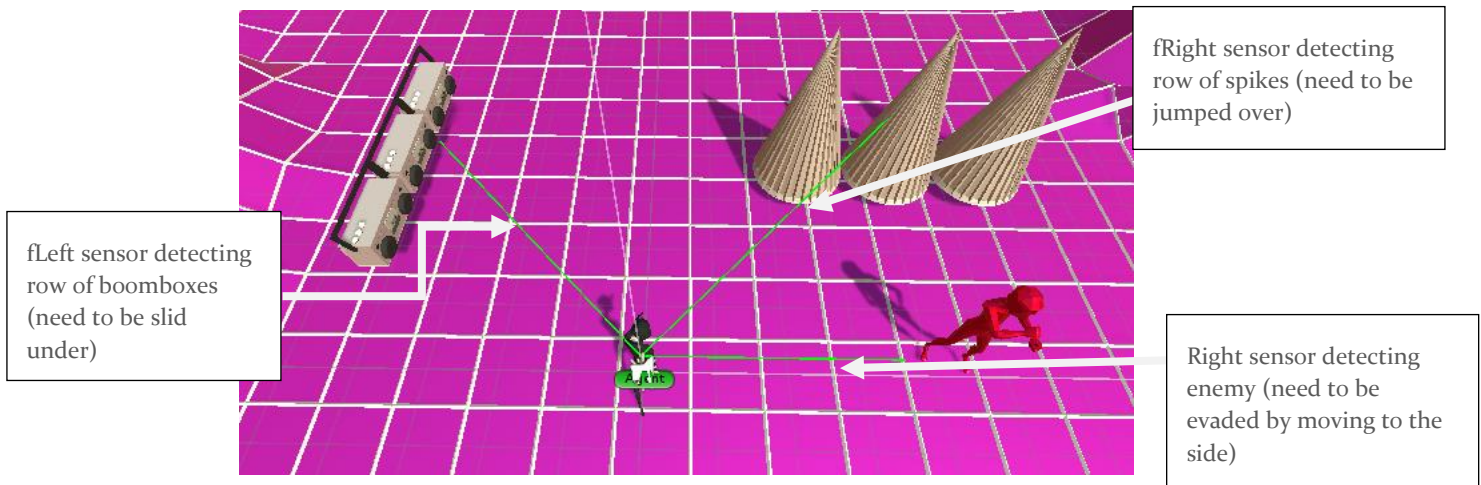


Figure 34: Sensors of one Agent detecting 3 different obstacles

Tests:

The only tests that need to be run for this are ensuring that all sensors are able to detect an obstacle, and transitively that all obstacles are detectable.

Tests 1 and 2: Ensuring all sensors are working and all obstacles are detected

- **Expected Output:** When an obstacle is put in its line of sight, a given sensor should be activated. Moreover, all obstacles should trigger an activation.

- **Test Output:**

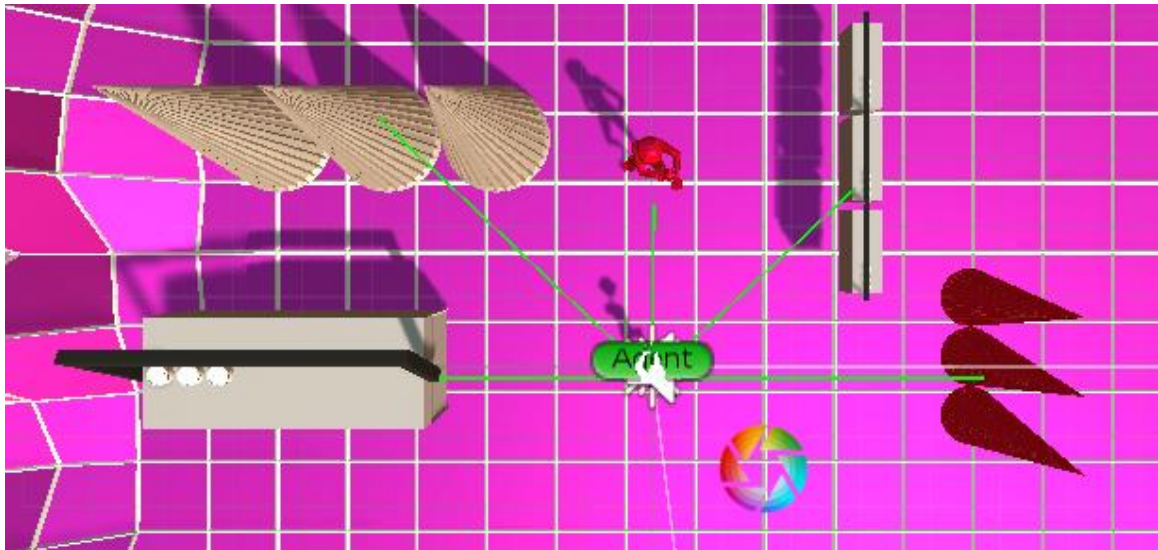


Figure 35: Obstacles put in sensors' line of sight

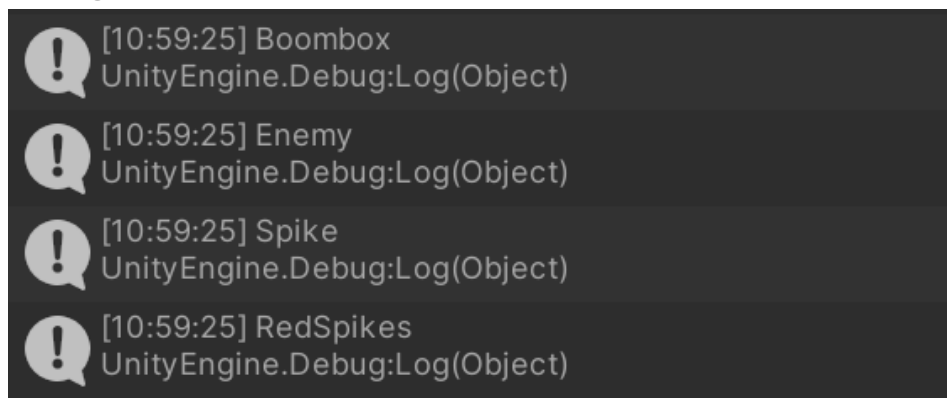
- **Test Outcome:** The two simultaneous tests were successful – every sensor is able to detect, and every obstacle is detectable.

Test 3: Ensuring sensors detect correct objects

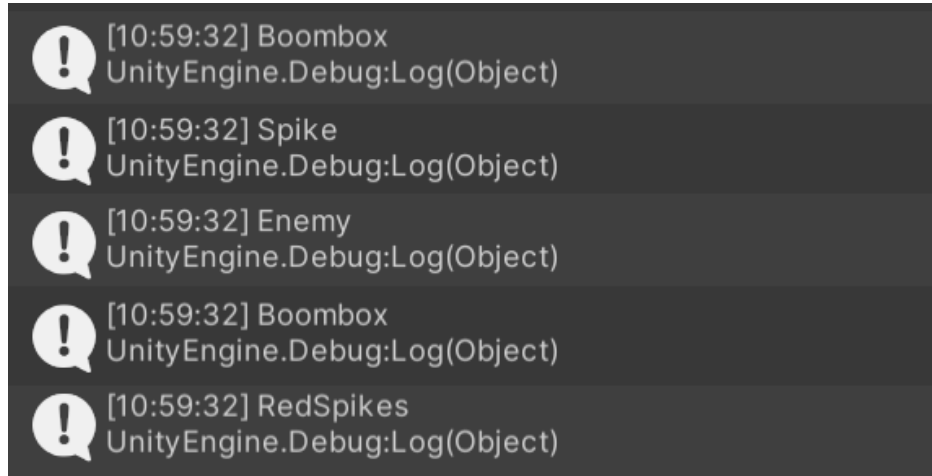
- **Overview:** In order for the agent's NN to work, the sensors need to correctly record the distance and tag of the object, as different distances and objects will likely result in different behaviour.

`Debug.Log(hit.collider.gameObject.tag);`

- **Expected Output:** Each sensor's distance collided object in the above image should be outputted to the Debug console
- **Testing Output:**



- **Test Outcome:** The output was not what I expected – I was expecting each individual object around the sensor to be outputted. I then learnt that the Unity Debug Console groups identical outputs under one output. I can simply toggle this off using a button. After that, I receive the expected output.



- **Refinements:** An addition that is pivotal to make is to add two more objects that can be considered 'obstacles'. These are the two invisible walls to the left and right of the agent. These were present in the base ER game to ensure the player did not veer off to sides of the course. I had removed them when initially updating my ER as I thought they would not be needed. I then implemented a new system that simply checks the x coordinates of the agent character to see if it is above or below certain values. I believe that, in the final solution, I will need to add the walls back. The overall functionality of the Sensors will not change as a result of this.

Neural Network Structure

Description:

The hyperparameter structure of my NN needs to be implemented in such a way that Unity can easily access and modify said hyperparameters. This will be done using MathNet's Matrix class. Moreover, it is probably better to not hard code the hidden layer structure and instead modularise the code by allowing the number of layers and nodes per layer to be defined in the Unity Editor. This is because, whilst the number of inputs and outputs tend to stay fixed when testing the effectiveness of an NN, the hidden layer architecture is often altered in hopes of better results.

Pseudocode:

```
SUBROUTINE InitialiseNN()  
  
    // Populate weight array with matrices
```

```

FOR i <- 0 TO numHiddenLayers
    IF i = 0
        // This is for the first set of weights, from the input layer to the
        // first hidden layer

        this.weights.add(Matrix(numHiddenNodes, numInputs + 1))
    ELSE IF i = numLayers - 1
        // Last hidden layer to output layer weights

        this.weights.add(Matrix(numOutputs, numHiddenNodes + 1))
    ELSE
        // Hidden layer to hidden layer weights

        this.weights.add(Matrix(numHiddenNodes, numHiddenNodes + 1))
    ENDIF
ENDSUBROUTINE

```

Models:

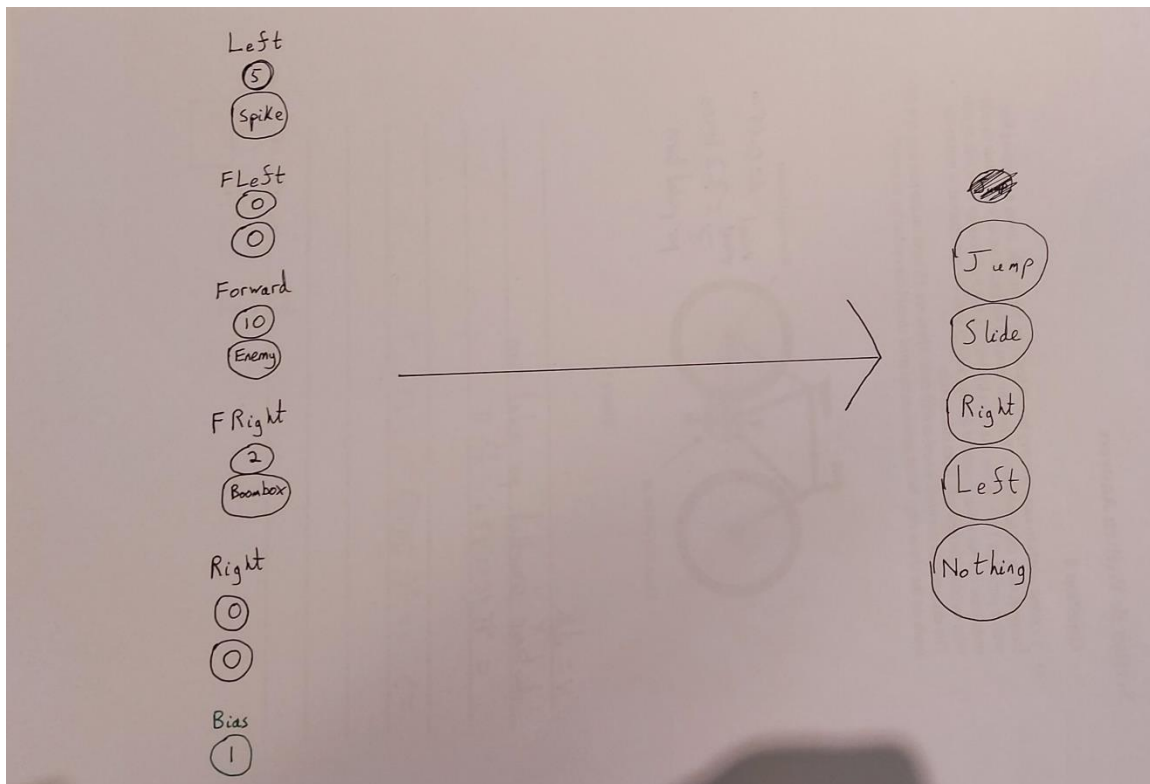


Figure 36: Abstracted model of input and output layers

In the above figure, the two inputs for each sensor are grouped in pairs, with the top node in each pair being the distance node (how far away the collided object is) and the bottom

one being the object node (which object was collided with). A value of 0 in the object node. The outputs are the five different actions that an agent can take.

In this model the object values are represented as strings of what the object actually is e.g Spike, but in the finished solution there will only be numerical values passed into the NN. The outputs will also be represented as numbers, which in turn represent the relative **probabilities that the corresponding move is the best one to make**. Thus, the network will choose to output with the highest value as its strongest prediction.

The hidden layers do not represent anything particularly meaningful at face value, but they are important as they determine the NN's capacity to learn. They can easily be represented as an array of matrices, with each element representing one layer – thus, for my network structure, the hidden layers array will have a size of 3.

Unity Prototype:

```
public class NN
{
    public int numHiddenNodes = 8;
    public int numHiddenLayers = 3;
    // 1 x 10 matrix as there are 10 inputs
    public Matrix<float> inputLayer = Matrix<float>.Build.Dense(1, 10);

    // There are a dynamic number of hidden layers so we need a list of matrices
    public List<Matrix<float>> hiddenLayers = new List<Matrix<float>>();

    // 5 outputs
    public Matrix<float> outputLayer = Matrix<float>.Build.Dense(1, 5);

    // Weights between layers
    public List<Matrix<float>> weights = new List<Matrix<float>>();

    // List of biases which corresponds to the bias of a layer
    public List<float> biases = new List<float>();

    public void Init()
    {
        // Clear all matrices of garbage values
        inputLayer.Clear();
        hiddenLayers.Clear();
        outputLayer.Clear();
        weights.Clear();
    }
}
```

3 hidden layers, 8
nodes per layer

0th element in
biases list is bias
for input layer, last
element is bias of
last hidden layer

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College.
Centre Number: 74615. Component Code: 7517/C.

```
        biases.Clear();

        // Populate hidden layers
        PopulateNN();
    }

public void PopulateNN()
{
    // Handle hidden layers and weights
    for (int i = 0; i < numHiddenLayers + 1; i++)
    {
        Matrix<float> newHiddenLayer = Matrix<float>.Build.Dense(1, numHiddenNodes);
        hiddenLayers.Add(newHiddenLayer);
        biases.Add(Random.Range(-1f, 1f));

        // Matrix for input to first hidden layer will have different dimensions
        if (i == 0)
        {
            Matrix<float> firstWeights = Matrix<float>.Build.Dense(10, numHiddenNodes);
            weights.Add(firstWeights);
        }

        if (i < numHiddenLayers - 1)
        {
            Matrix<float> newWeights = Matrix<float>.Build.Dense(numHiddenNodes, numHiddenNodes);
            weights.Add(newWeights);
        }
    }

    // Matrix for last hidden to output layer
    Matrix<float> outputWeights = Matrix<float>.Build.Dense(numHiddenNodes, 5);
    weights.Add(outputWeights);
    biases.Add(Random.Range(-1f, 1f));
}
```

Add a new hidden layer for each count of hidden layer count, and add a corresponding bias for that layer

For two connected layers with x and y number of neurons respectively, the weight matrix between them will have dimensions x by y , not including bias terms – biases are handled in their own list as during the Feed Forward stage, so they need to be treated slightly differently.

```
public void RandomiseWeights()
{
    for (int i = 0; i < weights.Count; i++)
```

WHOLE matrix is populated with random values on NN init

```
{
    for (int j = 0; j < weights[i].RowCount; j++)
    {
        for (int k = 0; k < weights[i].ColumnCount; k++)
        {
            weights[i][j, k] = Random.Range(-1f, 1f);
        }
    }
}
```

MathNET matrices are indexed as such

Tests:

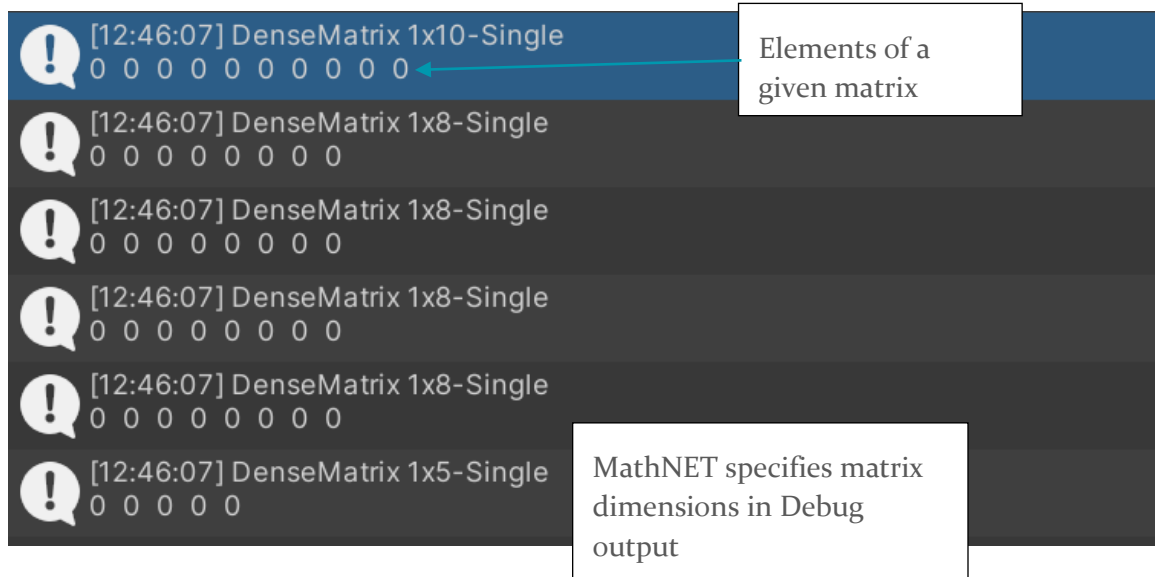
Test 1: Ensuring correct structure of NN

- **Overview:** The structure of my proposed NN in my [Analysis Modelling](#) should be held. That means that there should be 10 input nodes, 3 sets of 8 hidden nodes, and 5 output nodes. The biases are handled separately.
- **Expected Outcome:** When I output the matrices of the input, hidden and output layers, they should all have the correct dimensions. I have created a test GameObject with an instance of an NN to verify this.

```
void Start()
{
    test = new NN();
    test.Init();
    Debug.Log(test.inputLayer);
    foreach(Matrix<float> m in test.hiddenLayers)
    {
        Debug.Log(m);
    }
    Debug.Log(test.outputLayer);
}
```

Display each layer, which is formatted nicely in matrix form

- **Test Output:**



- **Test Outcome:** Whilst the dimensions of each layer are correct, there are one too many hidden layers. This was due to me accidentally looping over the specified hidden layer count one too many times. The corrections are shown below:

```
// Handle hidden layers and weights  
for (int i = 0; i < numHiddenLayers; i++)  
{
```

Loop numHiddenLayers times not numHiddenLayers + 1 times.

Tests 2 and 3: Ensure weight matrices are correct dimensions and contain random values

- **Overview:** On initialisation, the weights of the NN can just have random values, as the GA is responsible for altering the weight matrices such that the NNs make optimal actions. In addition, it is required that the weights are of a specific dimensionality relative to the two layers that they connect. This is because the matrix multiplication of the weights with the preceding layer needs to be a valid matrix multiplication (number of columns of M_1 = number of rows of M_2), and the resulting matrix needs to be of the dimensions of the next layer.
- **Expected Outcome:** I will initialise a test NN three times and output all four of its weight matrices each time. These should all be filled with random values. Moreover, the weight matrices should be of dimensions 10x8, 8x8, 8x8 and 8x5, in that order. (Note: in the Test Output, all matrix values are not shown).

- **Test Output:**

```
[17:40:59] DenseMatrix 10x8-Single
-0.0723244  0.784356  0.714843  -0.618342  -0.660897  ..  -0.561372  -0.106239

[17:40:59] DenseMatrix 8x8-Single
-0.607716  0.259599  0.284065  -0.968187  -0.599227  ..  -0.165888  0.249583

[17:40:59] DenseMatrix 8x8-Single
-0.135487  -0.339828  0.0617646  0.206623  -0.632482  ..  0.426932  0.0769364

[17:40:59] DenseMatrix 8x5-Single
-0.883332  -0.302988  0.586606  -0.686565  -0.94251

[17:54:07] DenseMatrix 10x8-Single
0.434428  -0.614723  0.264842  -0.14928  0.874737  ..  0.121269  -0.882372

[17:54:07] DenseMatrix 8x8-Single
-0.900805  0.829504  0.912185  0.202099  0.521263  ..  0.85625  0.147655

[17:54:07] DenseMatrix 8x8-Single
-0.668398  0.571216  -0.439062  -0.61373  0.735407  ..  -0.91542  -0.76275

[17:54:07] DenseMatrix 8x5-Single
-0.923219  0.360835  0.615783  -0.744858  -0.34858

[18:45:38] DenseMatrix 10x8-Single
0.914181  -0.140464  0.23983  -0.172122  -0.60769  ..  0.979092  -0.644629

[18:45:38] DenseMatrix 8x8-Single
0.934339  0.5731  0.697451  0.318012  0.0572015  ..  0.858858  0.927641

[18:45:38] DenseMatrix 8x8-Single
0.373019  0.612313  -0.9664  0.548321  -0.114311  ..  0.694791  -0.769242

[18:45:38] DenseMatrix 8x5-Single
-0.670938  0.694916  0.427105  -0.485856  -0.916756
```

- **Testing Outcome:** The tests worked as expected – all values are random, between -1 and 1 and the matrices are all of correct dimensions.

Activation Functions

Description:

The values of each node in an NN are mapped to specific ranges determined by the activation function used for that layer. This allows for outputs to be normalised, saving memory and allowing for faster feed forwards. The functions essentially allow the NN to learn more complex behaviours from the data.

The two activation functions I need to employ are the Sigmoid function (usually represented by the Greek letter Sigma σ) and the Hyperbolic Tangent function (normally represented as \tanh , pronounced 'th-an'). The \tanh function maps inputs between -1 and 1, whereas the sigmoid function maps between 0 and 1. Because of this, the Sigmoid function is good for mapping values to probabilities, and thus I will use it for the output layer. The \tanh function is better for the hidden layers, as the sign of each node's value more strongly influences the mapped value.

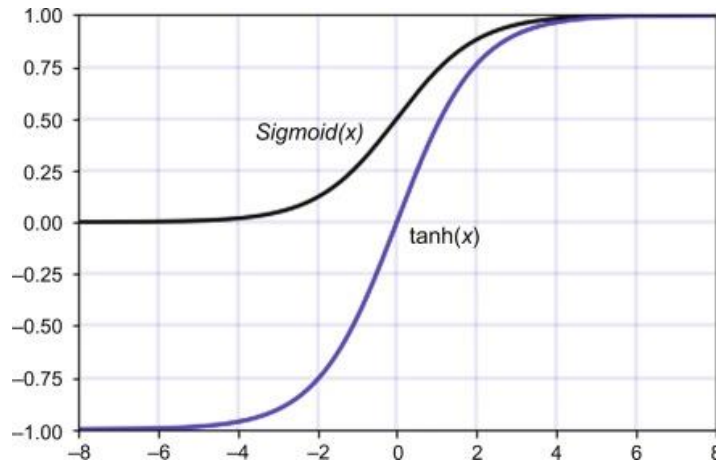


Figure 37: Graph of Sigmoid Function vs. Graph of \tanh Function

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \quad \sigma(x) = \frac{1}{1 + e^{-x}}.$$

Figure 38: Sigmoid and \tanh formulas

MathNet.Numerics, the C# Library I'm using, has a built-in \tanh function that works with matrices. Thus, I only need to implement my own Sigmoid method.

Pseudocode:

```
SUBROUTINE Sigmoid(z)
    // e = Euler's Number (2.718...)
    RETURN 1 / (1 + e-z)
ENDSUBROUTINE
```

Unity Prototype:

```
private float Sigmoid(float z)
{
    return (1 / (1 + Mathf.Exp(-z)));
}
```

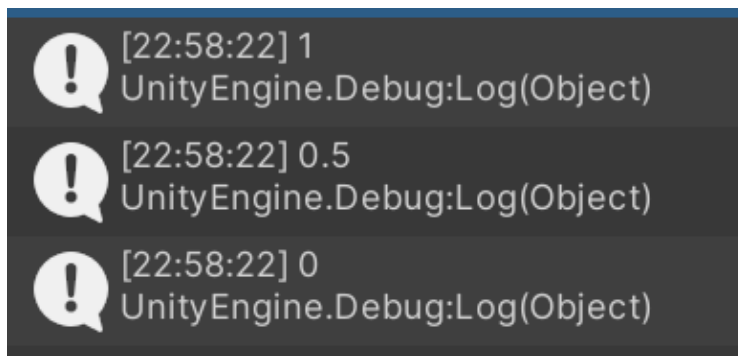
Tests:

Test 1: Ensuring common values of Sigmoid function are correctly calculated

- **Overview:** The C# compiler will round the Sigmoid of positive and negative values with high magnitudes to 1 and 0 respectively. Sigmoid(0) will return 0.5
- **Expected Outcome:** 1, 0.5 and 0 should be displayed to the Debug console, in that order.

```
void Start()
{
    test = new NN();
    test.Init();
    Debug.Log(test.Sigmoid(1000f));
    Debug.Log(test.Sigmoid(0f));
    Debug.Log(test.Sigmoid(-1000f));
}
```

- **Test Output:**



- **Testing Outcome:** The test ran as expected – the common values of the Sigmoid function were correctly outputted.

Feed Forward

Description:

In order to map inputs to outputs, NNs need to be able to translate their inputs forward through the layers that come after, eventually producing outputs in the output layer. This is done via feeding forward (at least in traditional NNs).

A Feed Forward method can be implemented via multiplying the matrices used to represent the network layers in series. The resulting vector will contain the outputs for that particular pass of the network. The Feed Forward method will likely run at least once per frame in the final solution, potentially even more if it is required to check it using Unity's Physics timestep system to improve how accurately agents can judge the optimal move with respect to time.

Pseudocode:

As mentioned above, the whole NN class has already been written in pseudocode. The FeedForward pseudocode was included in this NN class. The FeedForward method is rewritten below

```
SUBROUTINE FeedForward( )  
  
    // Activation functions will be discussed in following headings  
  
    this.inputs <- Tanh(this.inputs)  
  
    // Matrix multiplication to get the activation of each node  
  
    this.hiddenLayers[0] <- Tanh(this.weights[0] * this.inputs)  
  
    FOR i <- 1 TO numHiddenLayers  
        this.hiddenLayers[i] <- Tanh(this.weights[i] * this.hiddenLayers[i-1])  
    ENDFOR  
  
    this.outputs = this.weights[numLayers-1] * this.hiddenLayers[numHiddenLayers-1]  
  
    // Output normalised between 0 and 1 as they will be probabilities that that  
    action is optimal  
  
    RETURN Sigmoid(this.outputs)  
  
ENDSUBROUTINE
```


Unity Prototype:

```
public void FeedForward()
{
    // Activate Input Layer
    inputLayer = inputLayer.PointwiseTanh();

    // Input to first hidden layer
    hiddenLayers[0] = (inputLayer * weights[0]) + biases[0];
    hiddenLayers[0].PointwiseTanh();

    // First to penultimate hidden layer
    for (int i = 1; i < hiddenLayers.Count; i++)
    {
        hiddenLayers[i] = ((hiddenLayers[i-1] * weights[i]) + biases[i]).PointwiseTanh();
    }

    // Last hidden layer to output layer
    outputLayer = ((hiddenLayers[hiddenLayers.Count-1] * weights[weights.Count-
1]) + biases[biases.Count-1]);
    for (int i = 0; i < outputLayer.ColumnCount; i++)
    {
        // Sigmoid every element in output vector
        outputLayer[0, i] = Sigmoid(outputLayer[0, i]);
    }
}
```

Add corresponding bias to
each layer.

Output Layer now contains probabilities
of a certain action being optimal.

As shown in the above code screenshot, we start with the input layer as a vector, perform the tanh function on it (which performs the function on each individual element), add the input layer's bias to it, and multiply the result with the first weight matrix. The resulting vector is the values of the first hidden layer. We then tanh this layer, add its bias and multiply this vector with the second weight matrix. The resulting vector is now the values of the second hidden layer. The same process repeats for the 3rd hidden layer and output layer, the only differences being that the output layer does not have a bias and takes the Sigmoid activation function rather than tanh.

Tests:

Test 1: Ensure output layer has values between 0 and 1

- **Overview:** If the Feed Forward process has worked correctly, the output layer should have been Sigmoided, and so we will have 5 values between 0 and 1

- **Expected Output:** I will feed forward the network 5 times and output the values in the output layer each time. All of the values should be floating point values between 0 and 1.
- **Test Output:**



```
[23:10:19] DenseMatrix 1x5-Single
0.427406 0.436771 0.426462 0.305214 0.474222
[23:13:20] DenseMatrix 1x5-Single
0.529743 0.505394 0.832174 0.598588 0.765225
[23:13:26] DenseMatrix 1x5-Single
0.895829 0.943835 0.868205 0.79779 0.450468
[23:13:32] DenseMatrix 1x5-Single
0.837435 0.515736 0.635839 0.314411 0.453566
[23:13:39] DenseMatrix 1x5-Single
0.714929 0.315044 0.294349 0.495189 0.742308
```

- **Test Outcome:** The test worked as expected – all values are in the correct range, meaning they can all represent probabilities.

Agent

Description:

Now that the NN has been fully designed and prototyped, it is time to attach an NN object to a single agent, link the Sensor System values to the object as inputs, and then feed forward to check if the agent follows the behaviour given to it by the NN's outputs. It is likely that, since the NN has random weights and has no concept of learning how to run yet (as the GA has not yet been tested), the agent's behaviour will be chaotic, and it will likely not reach a survival time of longer than a few seconds.

I will need to map the output nodes of the NNs to one of the 5 actions an agent can take : left, right, jump, slide and nothing.

Pseudocode:

```
SUBROUTINE InitialiseAgent()  
    Instantiate new Agent  
    Initialise Sensor System of Agent  
    Initialise NN of Agent with random weights  
    Agent.NN.inputLayer = Matrix formed from Sensor System values  
    Render Agent
```

ENDSUBROUTINE

SUBROUTINE AgentAction()

Feed Forward agent's NN with current inputs to get outputs

a -> Action corresponding to output node with highest activation

Perform Action

ENDSUBROUTINE

Unity Prototype:



Figure 39: Agent GameObject in Hierarchy

// This method sets the input layer of the agent's NN to the values of the sensor system

```
void LinkToInputLayer(List<float> hits, List<int> collisions)
{
    Matrix<float> newInputs = Matrix<float>.Build.Dense(1, cols);

    int hitInd = 0;
    int colInd = 0;
    int ind1 = 0;
    int ind2 = 1;

    // Populate with sensor vals
    for (int i = 0; i < cols; i++)
    {
        if (i % 2 == 0)
        {
            newInputs[0, ind1] = hits[hitInd];
            ind1 += 2;
            hitInd++;
        }
        else
```

Inputs for one sensor are in pairs, so there is one odd and one even index. This is a rather crude way to do it, and will be refined in the actual implementation

```
{
    newInputs[0, ind2] = (float)collisions[colInd];
    ind2 += 2;
    colInd++;
}

agentNN.nn.inputLayer = newInputs;
// Debug.Log(newInputs);
}
```

I added the method above to the SensorSystem object's script. It runs every frame alongside the script that handles the sensors – every frame, the NN's input layer is updated based on the sensor values.

```
public class AgentNN : MonoBehaviour
{
    public NN nn;
    public PlayerMove movement;
    void Awake()
    {
        nn = new NN();
        nn.Init();
    }
    void LateUpdate() {

        // Choose max output

        for (int i = 0; i < nn.outputLayer.ColumnCount; i++)
        {
            if (nn.outputLayer[0, i] > maxOutput)
            {
                maxOutput = nn.outputLayer[0, i];
                maxOutputInd = i;
            }
        }

        // Choose max output

        switch(maxOutputInd)
        {
```

Reference to this agent's movement script so that the NN can influence its behaviours

This script is within the container for this agent's individual NN

This Unity function runs every frame but AFTER Update(). We need this here because the inputs are linked to the sensor in Update().

Simple search of index of max element in output vector

This next section maps index of the output vector to different actions. These actions are performed by accessing properties in the agent movement script from the base game's code

```
// LEFT (A key)
case 0:
    movement.horizontal = -1f;
    movement.isJumpPressed = false;
    movement.isSliding = false;
    movement.slideStopped = true;
    break;

// RIGHT (D key)
case 1:
    movement.horizontal = 1f;
    movement.isJumpPressed = false;
    movement.isSliding = false;
    movement.slideStopped = true;
    break;

// DO NOTHING
case 2:
    movement.horizontal = 0f;
    movement.isJumpPressed = false;
    movement.isSliding = false;
    movement.slideStopped = true;
    break;

// JUMP (Spacebar)
case 3:
    if (jumpCoolDownTimer == 0)
    {
        movement.horizontal = 0f;
        movement.isJumpPressed = true;
        movement.isSliding = false;
        movement.slideStopped = true;
    }
    break;

// SLIDE (S key)
case 4:
    movement.horizontal = 0f;
    movement.isSliding = true;
    movement.isJumpPressed = false;
    break;
}
}
```

This attribute controls the horizontal direction of the agent through its movement script. -1 = left, 1 = right

With this Agent object implemented, the following steps will occur each frame:

- Sensors are handled (hit distance and collision tags are extracted from sensors)
- These values are passed to NNs input layer
- NN feeds forward
- The index of the largest output node is found
- The corresponding action is carried out based on this index

Tests:

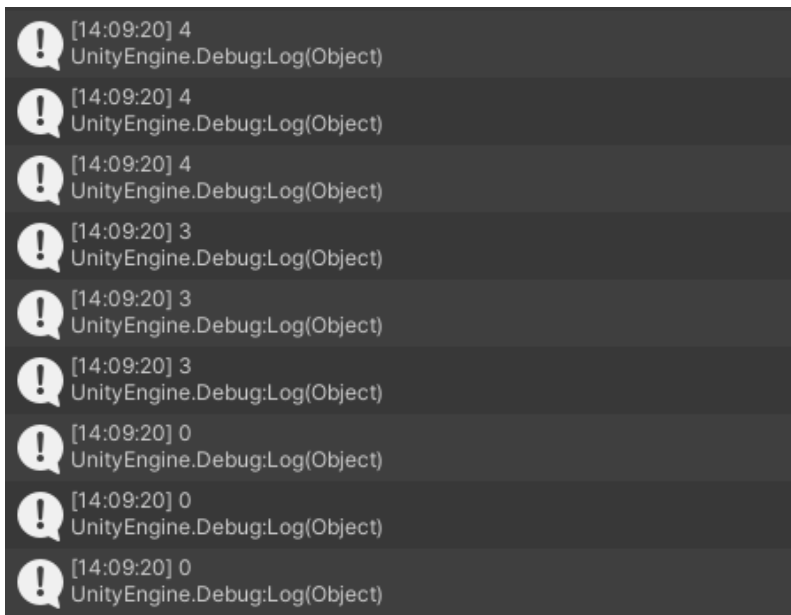
The only way sound way to test that this agent system is functioning as expected is just to verify that the action that the agent takes corresponds with the max output index that maps outputs to actions.

Test 1: Ensuring performed actions correspond to max output index

Expected Outcome: The agent's current performed action should correspond to the index of the output vector that maps to that action. The output index will be debug outputted. This means that:

- If the agent moves left, 0 is outputted
- If the agent moves right, 1 is outputted
- If the agent is not doing anything, 2 is outputted
- If the agent is jumping, 3 is outputted
- If the agent is sliding, 4 is outputted

Test Output:



In this particular run of the agent, it was clearly seen that it started off by sliding, followed by a slide and a slight move to the left, resulting in the agent hitting a spike and dying.

The ability of the NN to rapidly change actions, much faster than that of a human, sometimes interferes with the internal physics of the game. For instance, if the agent is constantly jumping, it actually manages to jump at slightly varying heights, which should not be allowed. I need to mitigate this somehow, likely by tweaking the movement and collision scripts of the game.

Test Outcome: This test was a partial success – whilst I can confirm that the system that I have created does indeed allow the agent's NN to influence it's movement, I have discovered other issues that hinder it. These will be addressed and fixed in the implementation, perhaps through some sort of cooldown system that limits the speed of actions.

Genetic Algorithm

The remaining components for this Design will be for the GA. Parent selection and crossover have not been prototyped as they are not core to the project's functionality – the algorithm can run without them, although it wouldn't be very effective

Population Initialisation

Description:

The last core component that needs to be added is the GA. To start, a population of the prototype agents need to be generated and rendered on the screen. I need to determine the max number of agents that Unity can comfortably handle without any major drops in performance, as this would likely translate into the finished build of the program. Moreover, there must always be at least 2 agents in the population, as otherwise the parent selection algorithm I will use will not work properly.

For the prototypes and tests, I will start by setting the max population size as 200 and will refine it to a more manageable amount if need be.

Pseudocode:

```
SUBROUTINE InitPop(popSize)
    FOR i IN RANGE(popSize)
        InitialiseAgent()
    ENDFOR
```

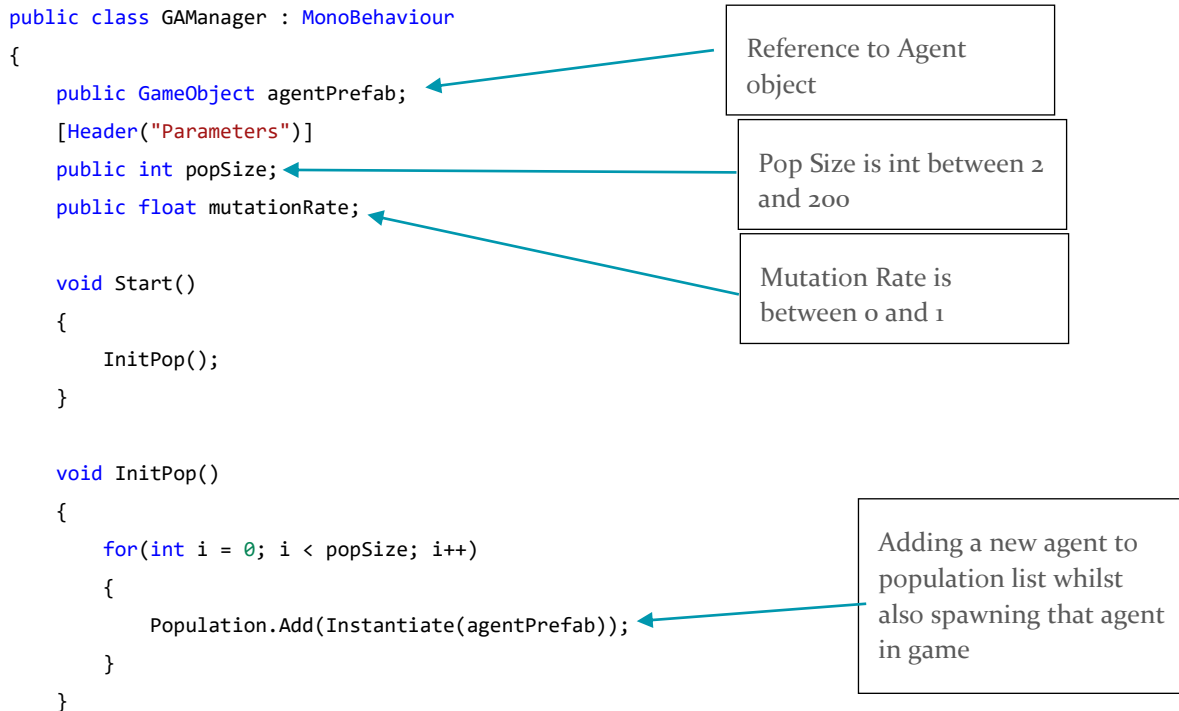
ENDSUBROUTINE

Unity Prototype:



This GameObject is for managing all elements of the GA

Figure 40: GameManager object in Hierarchy



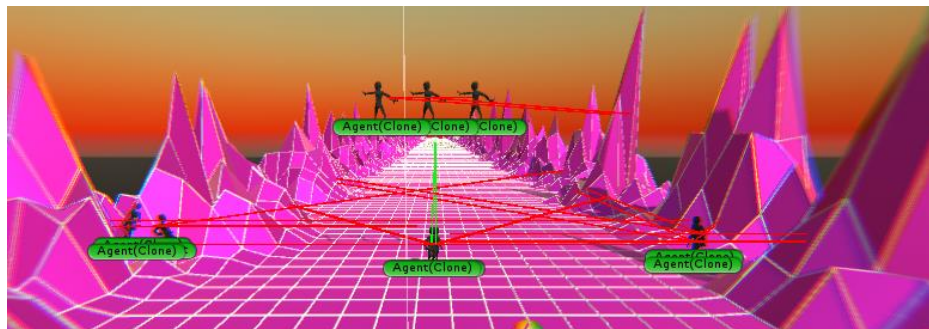
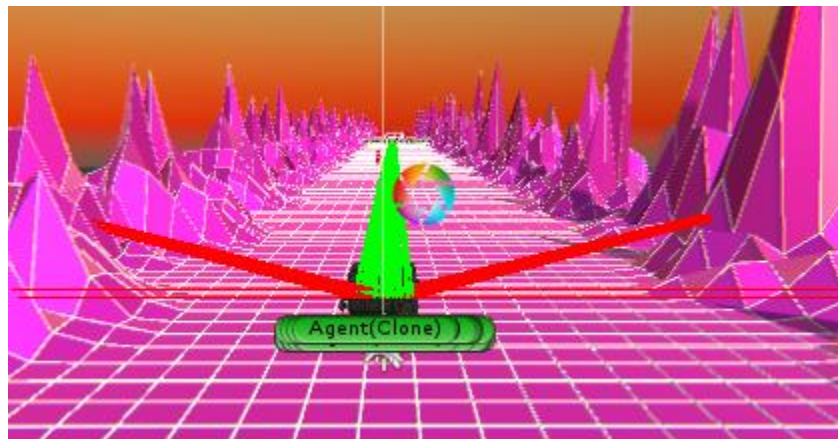
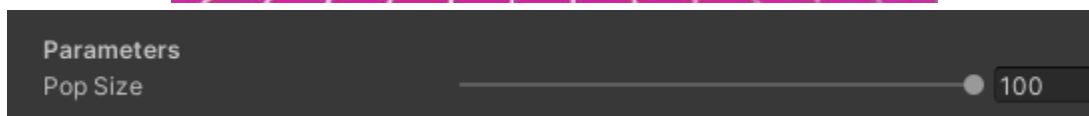
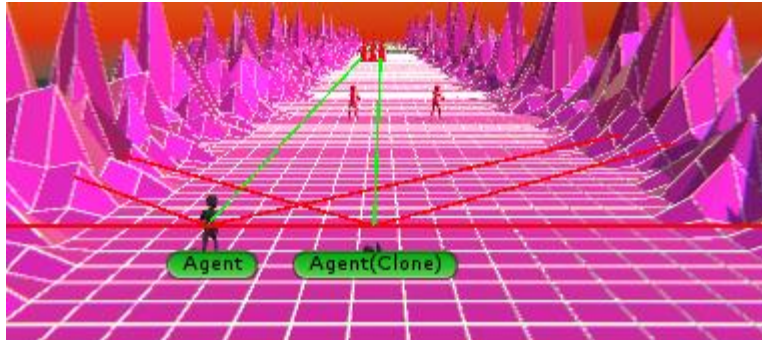
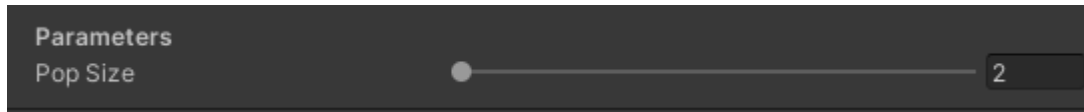
Upon first running this, I found that 200 agents rendered at once, all with their own Sensor Systems and NNs, was too much for the CPU to handle. As such, I will need to scale my maximum population down. 100 agents still run decently despite the amount of calculations happening every frame, so I will cap it at that. Hence, some of the specified values in my requirements may change e.g., Requirement 3a.

Moreover, the actual **rules** of the game need to be altered. The current GameManager script only accounts for one agent, so it needs to be modified to handle the deaths of multiple agents, as well as the ways in which those agents can die. This involves making minor edits to the base game code.

Tests:

Test 1: Ensure correct number of agents are spawned according to popSize

- **Overview:** Unsurprisingly, the number of agents specified by the GA parameter popSize needs to be the number of agents spawned.
- **Expected Outcome:** As stated above
- **Test Output:**



Figures 41-46: Testing Population Initialisation for various population sizes

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College.
Centre Number: 74615. Component Code: 7517/C.

Test Outcome: These tests all worked as expected – the correct number of agents spawned each time. However, the framerate dipped for 100 agents , so I need to limit the boundary of population sizes that can be selected. **This leads to me change Requirement 3a to allow a maximum population size of 50 rather than 100.**

Fitness Function

Description:

The fitness function for this particular application of a GA is, at face value, very simple; it is a viable approach for each agent's fitness score to be equal to the distance they travelled before dying (which I have defined as the score in the base ER prototype). The major drawback with this is that all agents will be misaligned with each other. Some of them colliding with the ground in a certain way that will misalign them in the Z-direction in 3D space. Thus, it is not wise to only use distance travelled as a metric for fitness. What I will instead do is, alongside the distance metric, add two new metrics, one that **counts how many times its corresponding agent has jumped, and the other doing the same thing for slides**. I can then set the values for each of these two new metrics to be the **reciprocal of the jump count and slide count**. Essentially, this penalises the agent for jumping/ sliding excessively. The rationale for this is that an agent in an early generation may have achieved a high distance simply by chance due to the fact that their NNs weight values made them constantly jump or slide (or both). In essence, a more effective agent would **perform actions only when they need to**.

The total fitness score will be a combination of all 3 metrics described. The formula can be described as such:

$$F(a) = d / j+s$$

Where the function F is the fitness function, a is an agent, d is the distance travelled by the agent a , and j and s are the jump and slide counts for a , respectively.

What this function does is decrease the **rate of increase of fitness the more the agent jumps/slides**. This means that there may still be a net increase in fitness, but another agent doesn't constantly jump will be able to gain fitness by travelling less distance.

Pseudocode:

```
SUBROUTINE Fitness(agent)
```

```
    RETURN agent.distanceTravelled / (agent.jumpCount + agent.slideCount)
```

```
ENDSUBROUTINE
```

Unity Prototype:

```
int jumpCount = 1;
int slideCount = 1;

// JUMP (Spacebar)
case 3:
    movement.horizontal = 0f;
    movement.isJumpPressed = true;
    movement.isSliding = false;
    movement.slideStopped = true;

    jumpCount++;

    break;

// SLIDE (S key)
case 4:
    movement.horizontal = 0f;
    movement.isSliding = true;
    movement.isJumpPressed = false;

    slideCount++;

    break;

public float Fitness(Transform agent, float jumpCount, float slideCount)
{
    return agent.transform.position.z / (jumpCount + slideCount);
}
```

These are initialised to 1 because in the fitness function, a division by 0 would equal infinity. It doesn't matter that they're not 0 as all the fitnesses of each agent are compared against each other with the same starting value

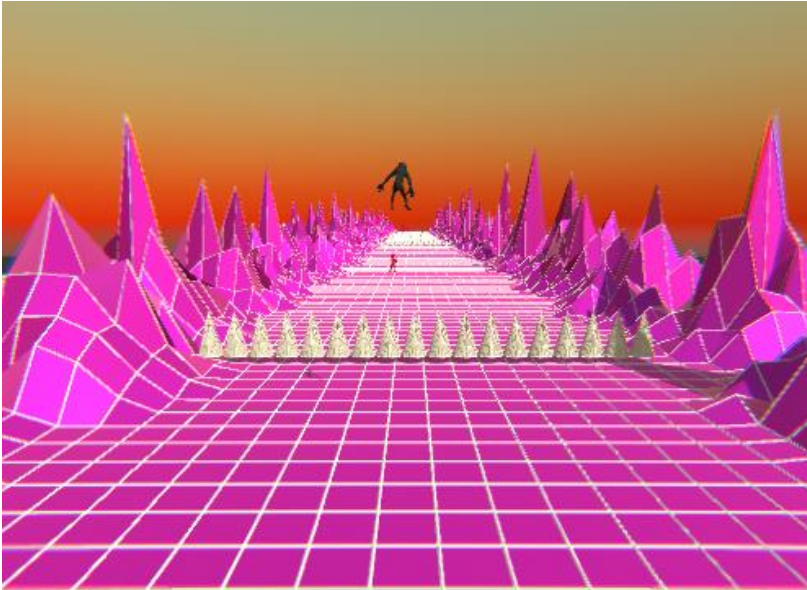
Tests:

There is no way for me to predict what the fitness of an agent will be, so I cannot have any exact expected outcomes for this test. Rather, I will simply continually output the fitness of one agent and see if its actions reflect the value.

Test 1: Check if fitness value reflects agent's behaviour

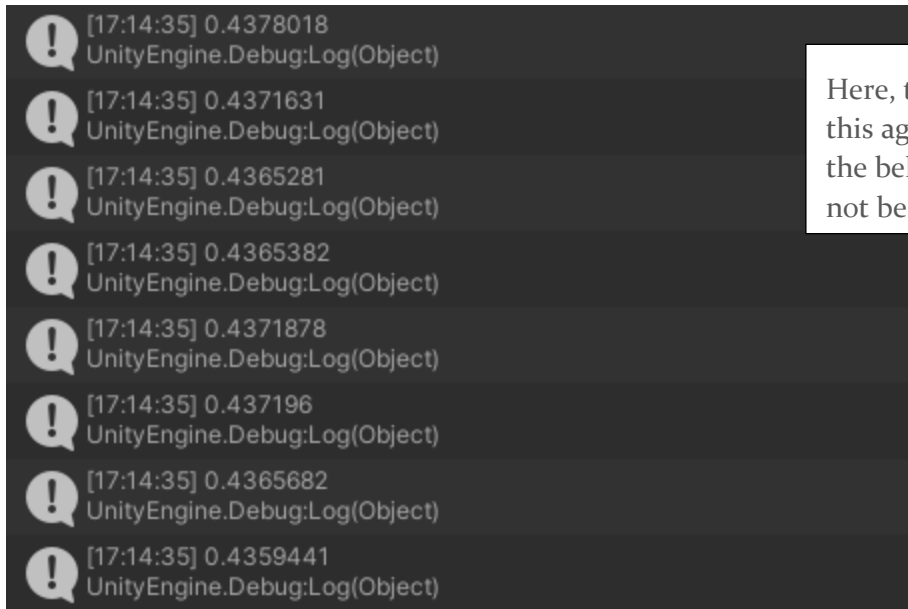
Expected Outcome: If the agent jumps or slides more frequently, it should gain fitness more slowly as opposed to if it stays still more often.

Test Output:

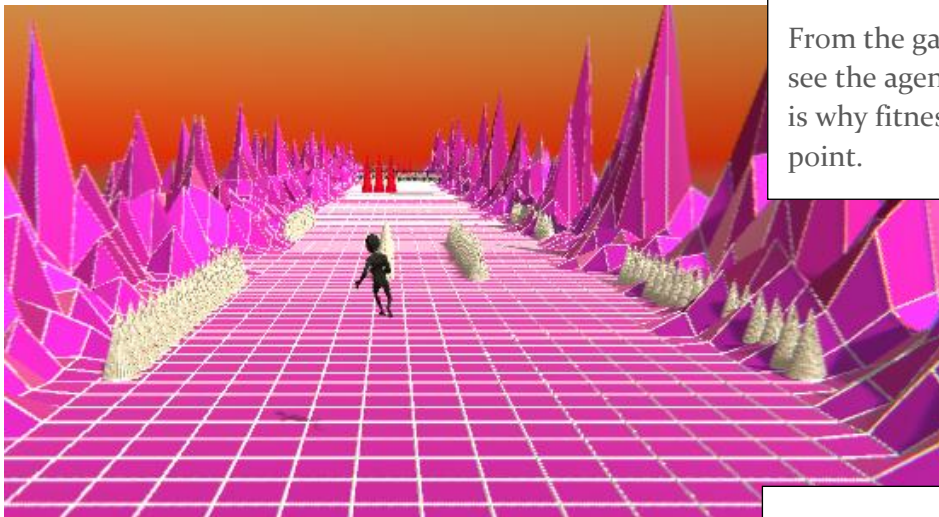


This agent was constantly jumping if its sensors were not detecting anything, so jumpCount was constantly increasing.

Figure 47: A test agent



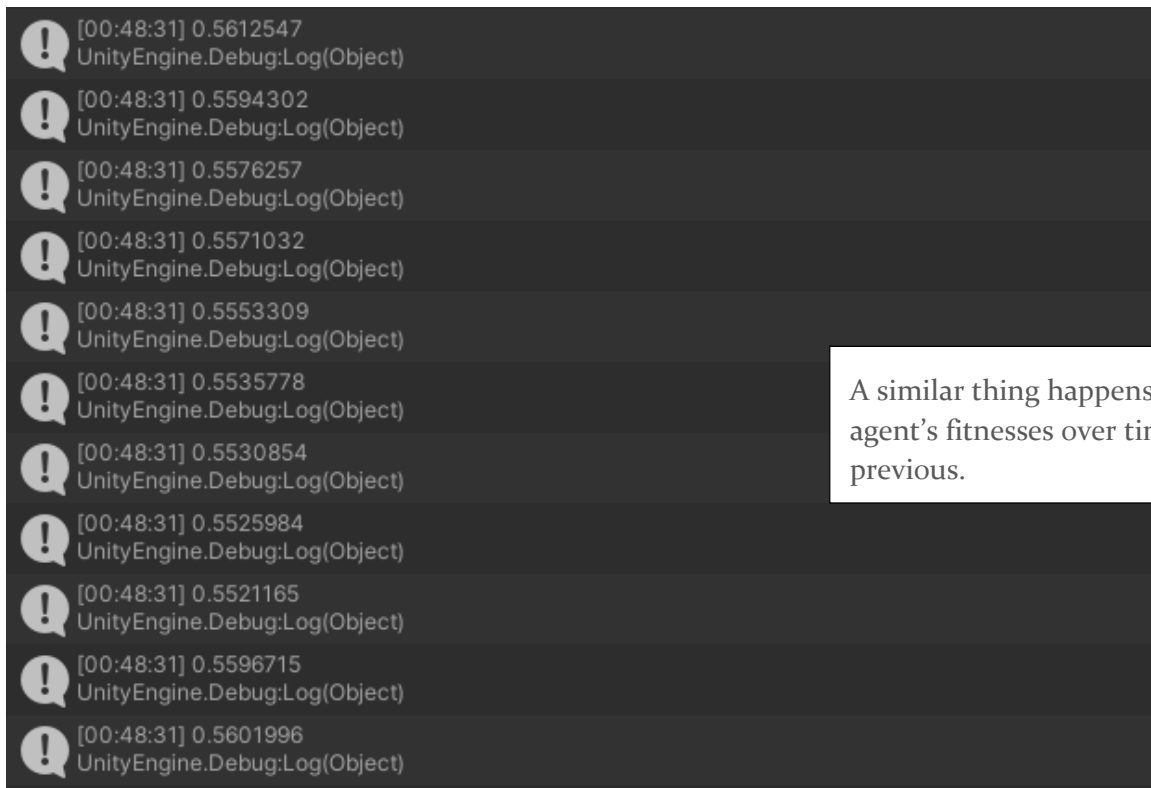
Here, the fitness is actually decreasing for this agent. This will greatly de-incetivise the behaviour of this agent, and so it will not be favoured by the GA



From the game window above, you can see the agent is about to stop sliding. This is why fitness begins increasing at this point.

Figure 48: Another test agent

The agent was in a constant slide state in this run.



Refinements:

After doing a few more test runs, I have noticed that the range of values that the fitness function can take is very large. This is because an agent that is jumping/sliding a lot is being divided by a very large amount compared to another agent. Whilst this is effective in showing that we do not want these types of agents in our population, it may cause problems in the Technical Solution phase when building a mating pool to select parents.

As such, perhaps altering the fitness equation may help with this issue. Something that doesn't divide the whole quantity but still shows that agents with higher jump/slide counts will have lower fitnesses.

$$F(a) = d - (1/j) - (1/s)$$

```
public float Fitness(Transform agent, float jumpCount, float slideCount)
{
    return agent.position.z - (1/jumpCount) - (1/slideCount);
}
```

This may be a better approach in terms of an equation. I will test this out on jumping agents, sliding agents and normal agents to see their variation in fitness.

After implementing this, I noticed that the fitnesses of jumping/sliding agents were almost as high as the normal fitnesses. However, the **gradient of fitness increase** was far lower (fitness increased less and less as time went on), so this feature was still retained from the previous equation. The higher fitness for these unwanted agents would mean that they are likely candidates for parents, which is not what we want. I either need to revert back to the previous fitness equation and normalise the fitness values, or switch to a new equation that does not need normalisation. I have decided to go with the former. The normalisation during the Parent Selection phase will be detailed in the Technical Solution.

Mutation

Description:

After agent reproduction (not shown in Design), the final stage of one pass of the GA is the mutation stage. In this stage, with some probability, the genetic information of a genome is randomly altered in some way. In this case, my mutation algorithm will take in one child NN as a parameter and then, with a probability of 5% (subject to change following tests), will set a random weight value in the weights to a random number within a specified range. This ensures that generations still maintain an element of uniqueness from the previous generation. The mutation function will be called on each child separately.

This is the final step of the GA. After the mutation process has occurred for some number of children such that the new generation now has the same population size as the current generation, then one cycle of the GA has been completed.

Pseudocode:

```
SUBROUTINE Mutate(child, mutationRate)

    r -> Random float from 0 to 1

    IF r <= mutationRate

        ind -> Random integer from 0 to child.weights - 1

        child.weights[ind] = Random float from -1 to 1
```

Unity Prototype:

```
void NN Mutate(NN nn)
{
    for (int i = 0; i < nn.weights.Count; i++) {
        float val = Random.Range(0f, 1f);
        if (val < mutationRate) {
            int rowInd = Random.Range(0, nn.weights[i].RowCount);
```



```
int colInd = Random.Range(0, nn.weights[i].ColumnCount);  
nn.weights[i][rowInd, colInd] = Random.Range(-1f, 1f);  
}  
}
```

Choose a random
weight in the
current weight
matrix

Corresponding weight
is randomised

Tests:

Tests 1 and 2: Ensuring values are being mutated at defined rate and that values are actually being mutated

Overview: Using a similar test NN to that shown in the tests for [Activation Functions](#), I will check whether mutation is working properly using the following code:

```
public class TestNN : MonoBehaviour  
{  
    NN test;  
    void Start()  
    {  
        test = new NN();  
        test.Init();  
    }  
  
    void Update()  
    {  
        FindObjectOfType<GAManager>().Mutate(test);  
    }  
}
```

Figure 49: TestNN.cs script for NN testing



5% mutation
rate

Figure 50: Mutation Rate Slider in Unity Editor

```
if (val <= mutationRate)
{
    Debug.Log("MUTATING");
    int weightInd = Random.Range(0, agentNN.weights.Count - 1);
    int rowInd = Random.Range(0, agentNN.weights[weightInd].RowCount);
    int colInd = Random.Range(0, agentNN.weights[weightInd].ColumnCount);
    Debug.Log(agentNN.weights[weightInd][rowInd, colInd]);
    agentNN.weights[weightInd][rowInd, colInd] = Random.Range(-1f, 1f);
    Debug.Log(agentNN.weights[weightInd][rowInd, colInd]);
}

else
{
    Debug.Log("NOT MUTATED");
}
```



Debug statements to show when and how a mutation has occurred.

Figure 51: Code for testing mutation rate functionality

Expected Outcome: When a mutation occurs of the test NN, the message “MUTATED” should appear, followed by two values: the original weight value and the mutated value. These should be in the range of -1 to 1. Moreover, these mutation events should occur roughly 5% of the time.

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College.
Centre Number: 74615. Component Code: 7517/C.

Test Outcome:



Test Outcome: The test worked as expected – the NN was mutated rarely enough to lead me to believe that the 5% rate did indeed work. Moreover, all values are in the correct range.

Refinements: Just to be completely certain, I will perform the same tests with varying mutation rates and compare the results.

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College.
Centre Number: 74615. Component Code: 7517/C.

20%:

[11:16:42] MUTATING
UnityEngine.Debug.Log(Object)

[11:16:42] 0.9875662
UnityEngine.Debug.Log(Object)

[11:16:42] 0.6959524
UnityEngine.Debug.Log(Object)

[11:16:42] NOT MUTATED
UnityEngine.Debug.Log(Object)

[11:16:42] MUTATING
UnityEngine.Debug.Log(Object)

[11:16:42] 0.7453878
UnityEngine.Debug.Log(Object)

[11:16:42] -0.5593972
UnityEngine.Debug.Log(Object)

[11:16:42] NOT MUTATED
UnityEngine.Debug.Log(Object)

[11:16:42] NOT MUTATED
UnityEngine.Debug.Log(Object)

[11:16:42] NOT MUTATED
UnityEngine.Debug.Log(Object)

[11:16:42] NOT MUTATED
UnityEngine.Debug.Log(Object)

[11:16:42] NOT MUTATED
UnityEngine.Debug.Log(Object)

[11:16:42] NOT MUTATED
UnityEngine.Debug.Log(Object)

[11:16:42] NOT MUTATED
UnityEngine.Debug.Log(Object)

[11:16:42] NOT MUTATED
UnityEngine.Debug.Log(Object)

[11:16:42] NOT MUTATED
UnityEngine.Debug.Log(Object)

[11:16:42] MUTATING
UnityEngine.Debug.Log(Object)

[11:16:42] 0.6959524
UnityEngine.Debug.Log(Object)

[11:16:42] -0.1198765
UnityEngine.Debug.Log(Object)

[11:16:42] NOT MUTATED
UnityEngine.Debug.Log(Object)

[11:16:42] NOT MUTATED
UnityEngine.Debug.Log(Object)

[11:16:42] NOT MUTATED
UnityEngine.Debug.Log(Object)

[11:16:42] MUTATING
UnityEngine.Debug.Log(Object)

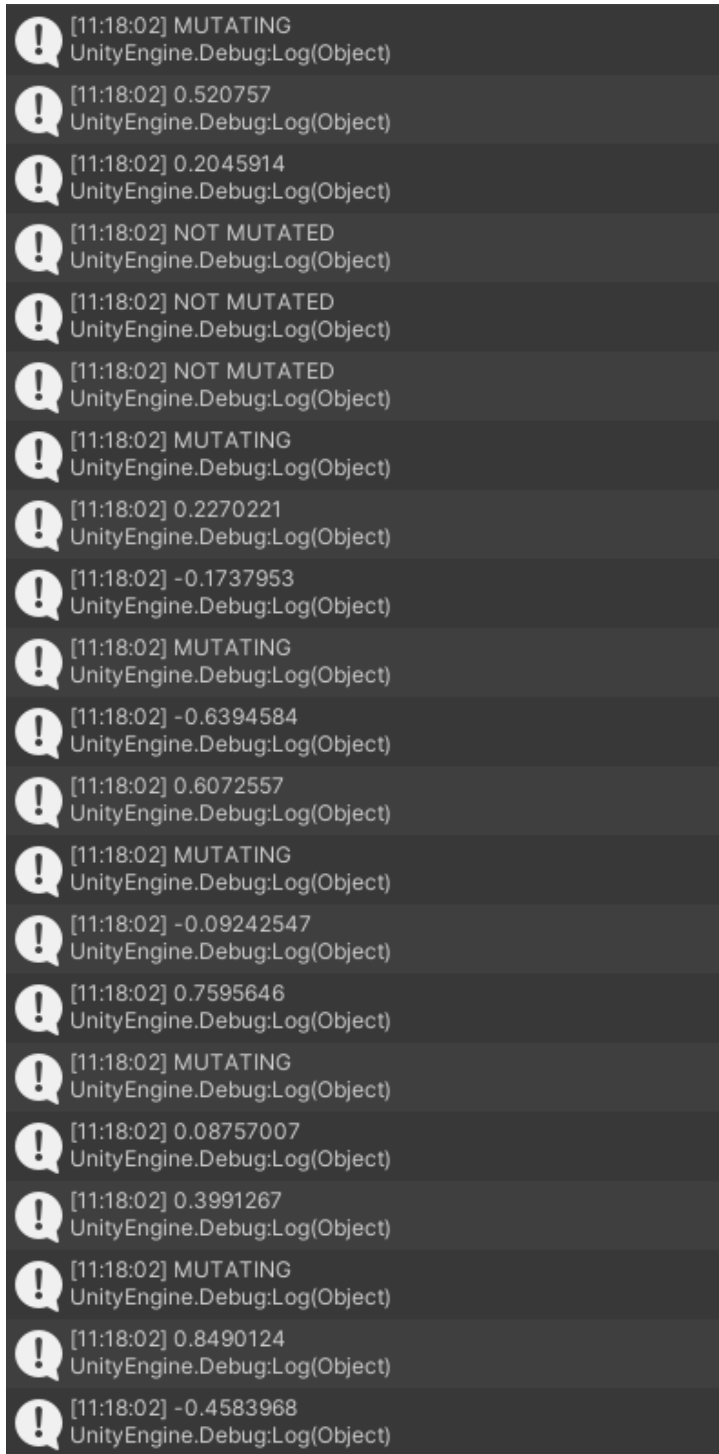
[11:16:42] -0.1515995
UnityEngine.Debug.Log(Object)

[11:16:42] 0.1970621
UnityEngine.Debug.Log(Object)

[11:16:42] MUTATING
UnityEngine.Debug.Log(Object)

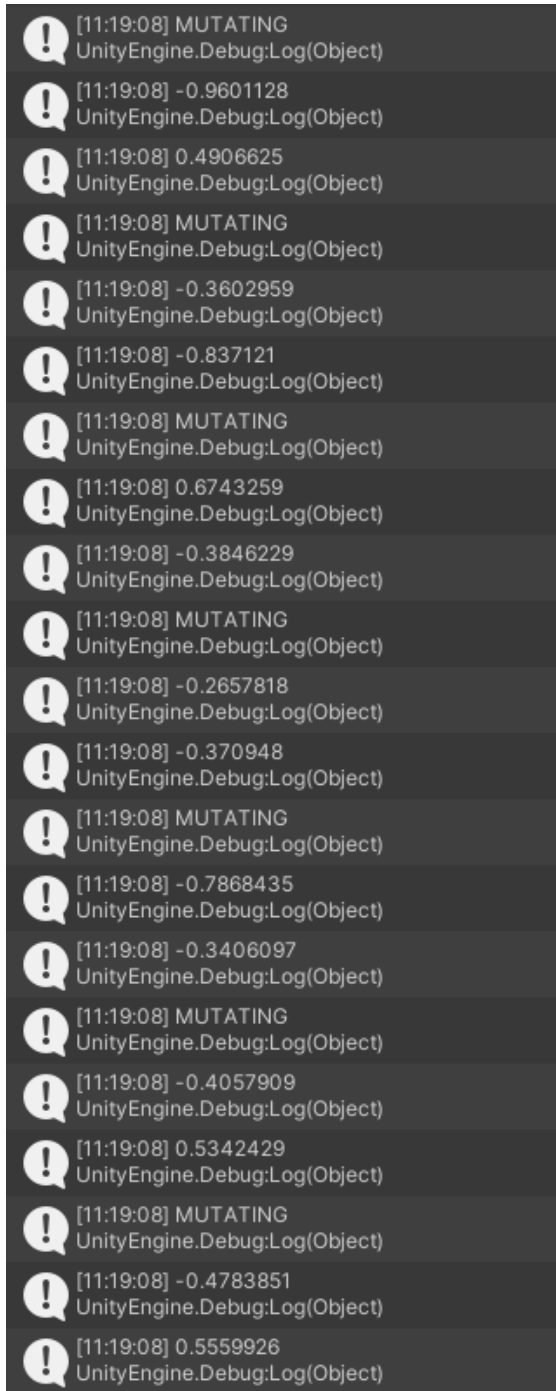
Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College.
Centre Number: 74615. Component Code: 7517/C.

50%:



Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College.
Centre Number: 74615. Component Code: 7517/C.

100%:



Test Conclusion: I am now certain that the mutation function works as expected.

Design Conclusion

The prototype has successfully worked. The core elements described earlier in this section have all been implemented and tested as intended. There were few test cases where there were unexpected results, but I have taken note of them and will mitigate/ make fixes to the issues that arose as a result of some componenets being prototyped. Moreover, the GA will need to be refined to include parent selection and crossover as intermediate steps before mutation.

It would also be wise to alter the object structure of my system in some areas, as this would make the game run more efficiently. For instance, there may be a method to reduce the number of for loops present in the NN operations as well as portions of the GA. Moreover, I will commit to maintain rigorous consistency in my coding styles. That is to say, I will ensure camel case is obeyed for all variable names, and otherwise follow standard conventions of Unity programming with C#.

As I have now developed a proof of concept for all core elements, I can begin to develop the final solution.

Technical Solution

The following three tables acts as an index for the technical complexities featured in my project.

Group	Model (including data model/structure)	Page number(s) evidenced in my NEA	Algorithms	Page number(s) evidenced in my NEA
A	Neural Network data structure (complex mathematical model)	pg. 133	Matrix multiplication	pg. 134
			Implementation and use of element-wise Sigmoid activation function on matrices. Use of hyperbolic tangent algorithm.	pg. 134-135
			Elementwise updating and handling of Matrix elements	pg. 124, 139
	Genetic Algorithm (complex mathematical model)	pg. 121	Dynamic game object instantiation and updating	pg. 123, 143
			Genetic operators (Fitness function, parent selection,	pg. 124-125

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College.
Centre Number: 74615. Component Code: 7517/C.

	SensorSystem data structure (complex scientific model)	Pg. 136	alternating crossover and mutation)	
			Probabilistic techniques: Weighted Random Selection	pg. 123
			Raycast Vector calculations	pg. 137
			Dynamically passing data from raycast sensors to neural network inputs.	pg. 138

Technical B Skills

Group	Model (including data model/structure)	Page number(s) evidenced in my NEA	Algorithms	Page number(s) evidenced in my NEA
B	Multi-dimensional collections e.g., Matrix (including Lists of other iterables e.g. a List of Matrices)	Throughout (example on pg.132)	Insertion Algorithms	pg. 124, 139
	Dictionaries	pg. 136	Simple and appropriate updating of dictionary/record values	pg. 137, 156
	Records	pg. 154		
	Simple OOP model	Throughout (examples at pg. 133)	Array Normalisation algorithm	pg. 127
			Generation of objects based on simple OOP model	Throughout (example on pg. 140)
			C# Properties with accessor methods.	Throughout (examples on pg. 132, 153)
			Static members	Throughout (2 examples on pg. 133)

Technical C Skills

Group	Model (including data model/structure)	Page number(s)	Algorithms	Page number(s)
-------	--	----------------	------------	----------------

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College.
Centre Number: 74615. Component Code: 7517/C.

		evidenced in my NEA		evidenced in my NEA
C	Single-dimensional arrays	Throughout (examples on pg. 75, 140)	Linear search for max element	pg. 140
	Appropriate choice of simple data types	Throughout	Arithmetic mean	pg. 126

The remaining tables are indexes for the coding styles and practices employed for this project. I have endeavoured to maintain a consistent coding style and format throughout the entire codebase and have commented appropriately where necessary.

Style	Characteristic	Page number(s) evidenced in my NEA
Excellent	Modules with appropriate interfaces.	Throughout (example on pg. 123)
	Loosely coupled modules – module code interacts with other parts of the program through its interface only.	Throughout (example on pg. 132)
	Cohesive modules	Throughout (examples on pg. 135, 143)
	Modules with common purpose grouped.	Throughout (example in any script) (Unity MonoBehavior framework is built for this)
	Defensive programming.	pg. 120, 124
	Good exception handling.	pg. 152
Good	Well-designed user interface	Multiple examples. See pg. 145
	Modularisation of code.	Throughout (example at pg. 139)
	Good use of local variables	Throughout (Unity does not use global variables).
	Minimal use of global variables	Throughout (See above)
	Managed casting of types	pg. 151-152
	Appropriate indentation	Throughout
	Self-documenting code.	Throughout (examples at pg. 128, pg. 142-143)
	Consistent style throughout (bracketing and indentations consistent depending on type of clause, private identifiers are lowercase whilst public methods are capitalised. Fields are always lowercase.)	Throughout
	Switch cases	pg. 140-141
Basic	Meaningful identifier names.	Throughout

		(example on pg. 128)
	Annotation used effectively where required.	Throughout
	Basic Validation Methods	pg. 151-152

Project Structure

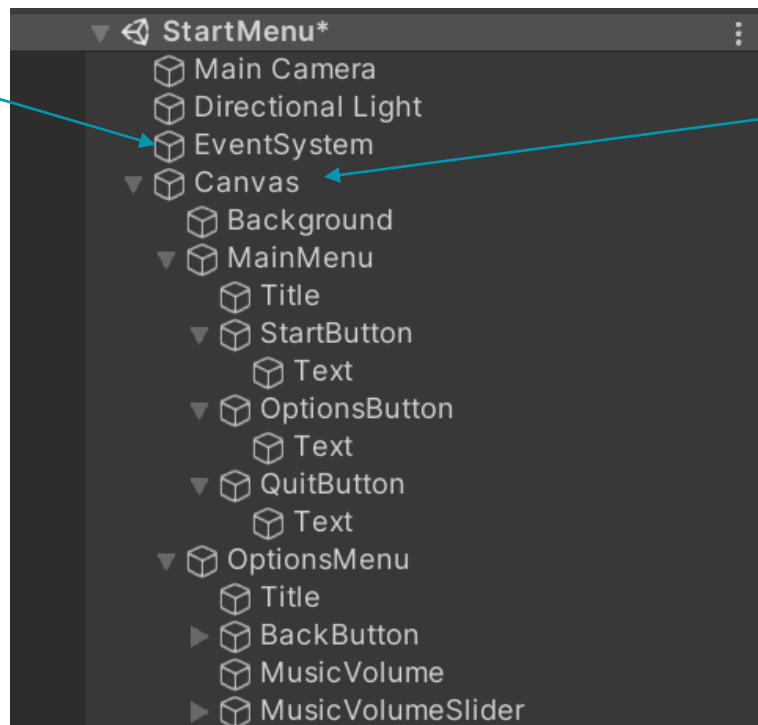
The structure of the Unity project can be shown in two different ways:

1. The Hierarchy of a given Unity scene, which shows all GameObjects as well as the GameObjects that are parented or childed to other GameObjects. Childed GameObjects inherit local position, rotation and scale from their parent (see [Prerequisites](#) for clarification on this).
2. Showing the file directory of the project's C# scripts which control the logic of the system in a text editor.

Scene Hierarchies

A Unity Scene is a game asset that contain a portion or all of a game's content. Each Scene is separate from oneanother, with their own GameObjects and UI. There are only two scenes in my project: the Start Menu Scene, which hosts the Main Menu, Options Menu, and GA Parameter Input Screen, and the Main Scene, which is where the main visualisation is hosted.

This GameObject is required for all Scenes where event handling is required (button presses, sliders, etc.)



Canvas is the object in which all the current Scene's UI is contained

Figure 52: "StartMenu" scene Hierarchy

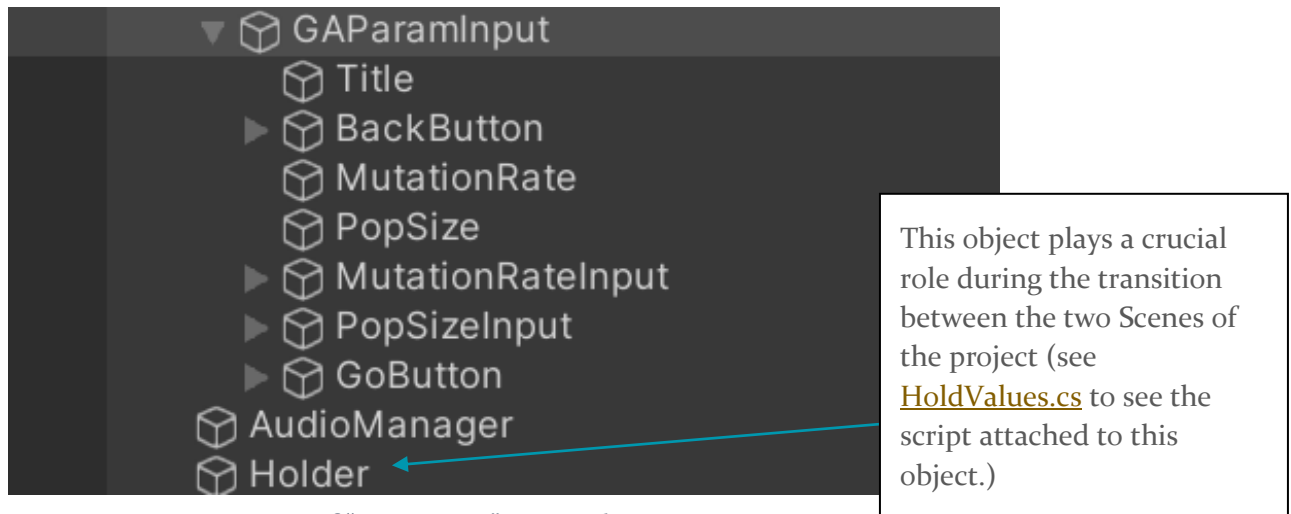


Figure 53: Continuation of “StartMenu” Hierarchy

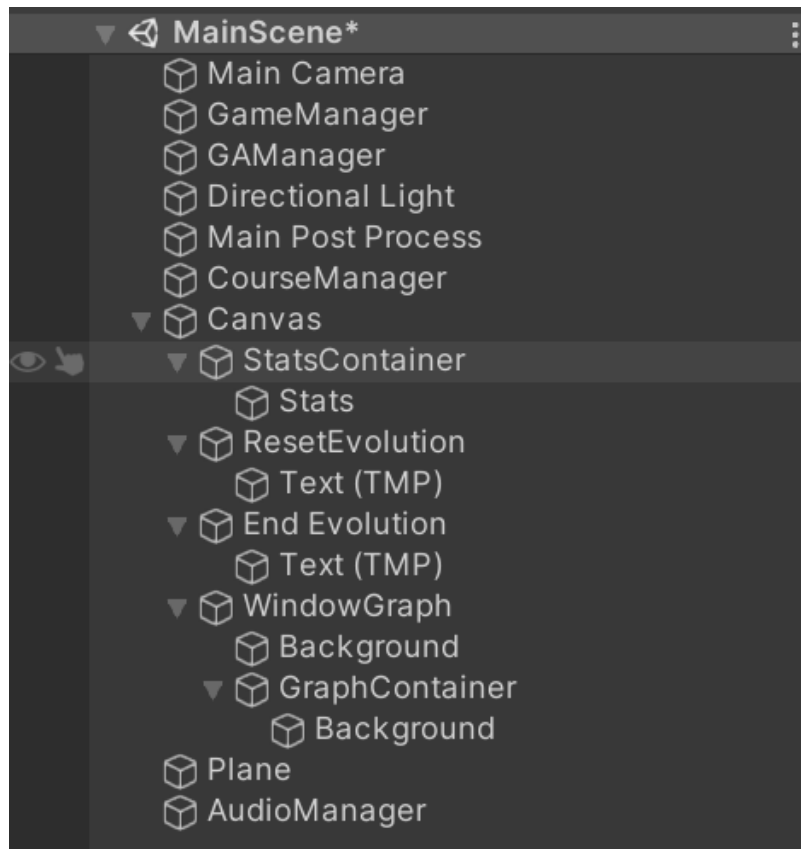


Figure 54: “MainScene” scene Hierarchy

File Directory

Below are all the scripts used in the project. Structure-wise, there was no need to further categorise the scripts into different sub-folders, as Unity allows any script to be integrated

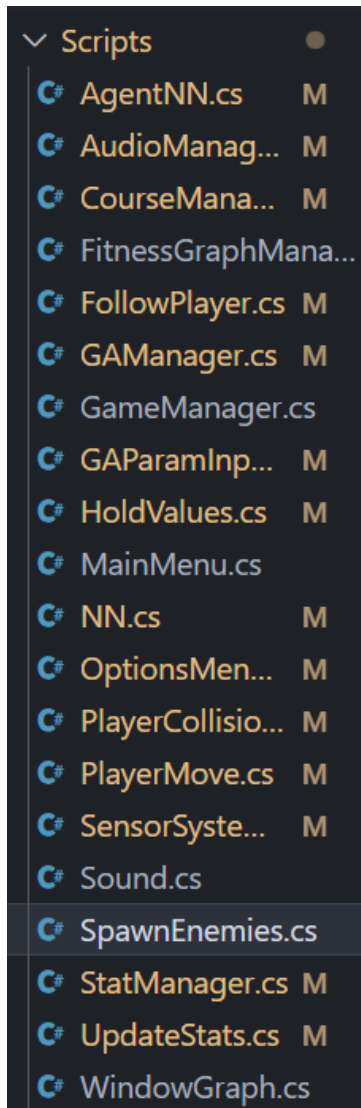


Figure 55: Project Files

with any object very simply and very easily. Additionally, some GameObjects might need multiple scripts, which may come from all different sub-categories, so this may make for more cumbersome design. Hence, this standard flat format was chosen. The scripts are contained within the Assets folder of any standard Unity project. Note that the yellow highlighting and other markings are simply a result of uncommitted changes to the GitHub repository that this project is stored in. Some of the scripts are ones that are from the original Summer 2020 ER prototype. These are AudioManager, CourseManager, FollowPlayer, DontDestroy, GameManager, OnSlide, PlayerCollision, PlayerMove, Sound, and SpawnEnemies. For some of these scripts, the code is simply overhead for the purpose of this project. However, changes were made to some of these scripts to accommodate for the new context of the project (multiple AI agents as opposed to one human player). The new scripts that were added with this project, as well as the scripts that had to be edited, will be documented throughout this section. For this writeup, I will aim to display and explain the scripts in as logical an order as possible, such that each one can be understood given the context of some of the previous ones, and vice versa; once a new script is shown, the previous ones make a little more sense. One final note is that I will be highlighting any major changes that have been made from the Requirements of my Analysis – there are not many of them, but the ones that are present are such because of unforeseen circumstances, predominantly due to the limitations of the

technology and platforms used. There is one that is better to explain in text rather than in the code: **Requirement 4d could**

not be met, as altering Unity's internal timescale too drastically would cause the system to crash due to too many calculations being performed per second. This is discussed further in my Evaluation.

Prerequisites

Before the codebase is documented here, there are a few Unity-specific idiosyncracies that need to be explained. All standard Unity scripts inherit from the UnityEngine namespace's MonoBehaviour class. This class is what allows the scripts to behave as they do in a Unity project. It also allows them to make use of special keyword methods that, depending on their name, will only call at certain times. These methods' names must be capitalised, regardless of protection level. As such, they transcend the formatting I have laid out in my codebase of private properties being lowercase. Examples of MonoBehaviour methods are the Start() method, which is called at runtime, whilst the Update() method is called every frame.

Below is what the framework of a simple Unity script would look like.

```
using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;
```

Allows us to use MonoBehaviour

```
public class SimpleScript : MonoBehaviour  
{  
    // Start is called before the first frame update  
    void Start()  
    {  
  
    }  
  
    // Update is called once per frame  
    void Update()  
    {  
  
    }  
}
```

Other **MonoBehaviour methods** will be pointed out when encountered.

Another fundamental concept in Unity development is the idea of components. Unity's component system can be thought of as a simplified visual OOP approach. Each GameObject is made up individual components; a simplified model of aggregation in OOP. There are many different types of components, each of which allows the GameObject to exhibit new behaviours or properties. The most basic component is the Transform, which every GameObject must have. It is what controls the position, rotation and scale of a GameObject in the world space. When we attach C# scripts to

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College.
Centre Number: 74615. Component Code: 7517/C.

GameObjects, **they are considered as components**. Thus, we can access references to C# scripts in the same way we access Transforms.

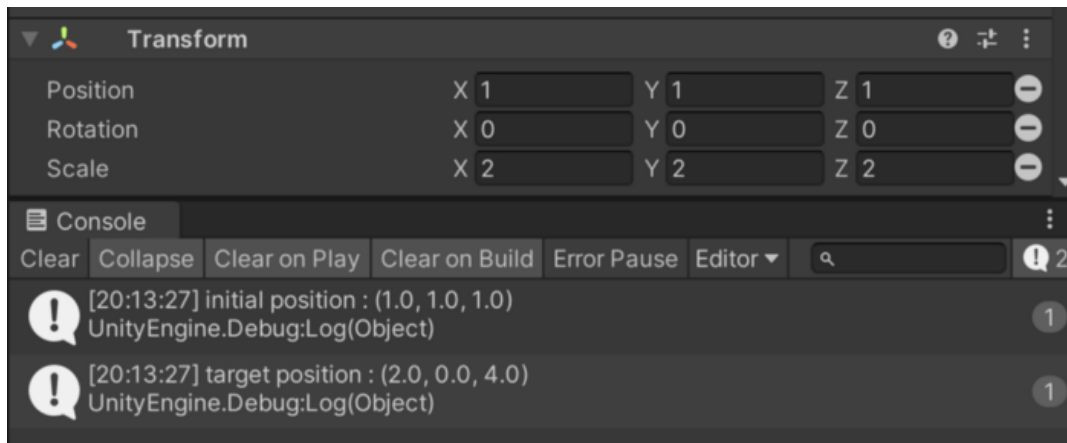


Figure 56: Example of a Transform component on GameObject

Components of any GameObject can be accessed and altered using the `GetComponent<>()` method. With this prerequisite knowledge, we can begin documenting the code.

GameManager.cs

This script controls how the MainScene runs. It handles when to restart the evolution by checking if all agents in the generation are dead, then calls the genetic algorithm's appropriate reset methods via a reference to the GAManager.cs script attached to the GAManager object. Similarly, GameManager.cs is attached to a GameManager object.

```
using UnityEngine;
using System;
using UnityEngine.SceneManagement;
using Random = UnityEngine.Random;
```

Unity's library that handles switching between scenes.

```
public class GameManager : MonoBehaviour
{
    private GAManager genetic;
    private WindowGraph graph;
    private HoldValues hold;
    public static GameManager instance;

    private void Awake()
    {
        if (instance == null) {
            instance = this;
        } else {
```

This script preserves variables within Scripts (see [HoldValue.cs](#)).

MonoBehaviour Method:
This is called when the Scene starts, before anything else.

This block of code is featured in a few other scripts – it is used in ensuring that an object is not destroyed when transitioning between the two Scenes in this project. Logic-wise, it ensures only one instance of the object is present in the Scene, and it is always the same instance.

```
        Destroy(gameObject);
        return;
    }
    DontDestroyOnLoad(gameObject);

    if (SceneManager.GetActiveScene().buildIndex == 0) {
        SceneManager.LoadScene("StartMenu");
    }

    genetic = GameObject.Find("GAManager").GetComponent<GAManager>();
    hold = GameObject.Find("Holder").GetComponent<HoldValues>();
    graph = GameObject.Find("WindowGraph").GetComponent<WindowGraph>();
}

private void LateUpdate()
{
    checkForRestart();
}

private void checkForRestart()
{
    // Checking if current generation has ended
    if (genetic.Population.Count == 0 && genetic.Initialising == false && !genetic.GameEnded) {
        Restart();
    }
}

public void Restart(bool resetEvolution=false)
{
    graph.AddNewPoint(genetic.AverageFitness());
    genetic.EndGeneration();
    graph.ClearGraph(resetEvolution);
    graph.ShowGraph();
    GameObject.Find("CourseManager").GetComponent<CourseManager>().SpawnFirstTiles();
    genetic.InitNewGen(resetEvolution);
    GameObject.Find("Main Camera").GetComponent<FollowPlayer>().ResetPos();
}

public void QuitToMenu()
{
    SceneManager.LoadScene("StartMenu");
    Destroy(gameObject);
}
}
```

This is the explicit method that signals that this object should not be destroyed on loading of a new scene.

Object should not be present in StartMenu scene. If it is, reload the scene. **Good Coding Practice:** Defensive Programming

MonoBehaviour Method: Called after all other Update methods. This is useful for events that need to happen after everything has rendered

This method handles resetting all of the elements of the Scene once the generation has ended.

Resetting all elements; handling new point on graph, resetting the camera position, starting the new generation of GA etc.

GAManager.cs

This is the script that implements everything to do with the genetic algorithm: Population initialisation, the Fitness Function, parent selection, crossover, and mutation are all in this script. These are all Technical A complexities because of their use of probabilistic techniques, such as a weighted random selection algorithm. Population Initialisation, Crossover and Parent Selection also dynamically handle neural networks as genomes. In addition, other genetic-related utility methods are included here, such as an AverageFitness method for the UI Graph (see in [WindowGraph.cs](#)) and an EndGeneration method. This is the largest, and arguably the most important script in the project. As such, most other scripts have been designed with this script in mind (if it is required for the two scripts to interact at one point or another). Note that many of the variables in this script are C# Properties, which are a flexible, compact mechanism for reading and writing to class members. They provide the easy access of public fields whilst also ensuring the safety and encapsulation of private fields. Since the entire system uses GA-related parameters a lot, these qualities were very useful.

```
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using MathNet.Numerics.LinearAlgebra;
using Random = UnityEngine.Random;
```

Linear algebra library required for incorporating Matrix data structures that make handling neural networks

```
public class GAManager : MonoBehaviour
{
    HoldValues hold;
    // This is a reference to a 'template' of an agent
    // Has player movement script, SensorSystem, NN etc.
    public GameObject agentPrefab;
    GameManager gm;
    public int GenerationNum
    { get; set; }

    [Header("Parameters")]

    public int popSize;
    public float mutationRate;

    public List<GameObject> Population
    { get; set; }
```

Technical A Skill: Genetic Algorithm (complex mathematical model)

Technical B Algorithm: C# Properties with correct accessor methods

Properties are formatted like this, in PascalCase and with { get; set; } following. This is a shorthand alternative for normal getter and setter methods.

```
// ith nn in the mating pool has the fitness at the ith position of the fitness list
```

```
public List<NN> MatingPool  
{ get; set; }  
public List<float> Fitnesses  
{ get; set; }
```

```
private List<NN> nextGen;
```

```
public bool Initialising
```

```
{ get; set; }
```

```
public bool GameEnded
```

```
{ get; set; }
```

```
public static GameManager instance;
```

```
void Awake()
```

```
{
```

```
    if (instance == null) {
```

```
        instance = this;
```

```
    } else {
```

```
        Destroy(gameObject);
```

```
    }
```

```
    DontDestroyOnLoad(gameObject);
```

```
    GameEnded = false;
```

```
    // GA Structures and Params
```

```
    MatingPool = new List<NN>();
```

```
    Fitnesses = new List<float>();
```

```
    Population = new List<GameObject>();
```

```
    nextGen = new List<NN>();
```

```
    Initialising = true;
```

```
    // Using the holder object that wasn't destroyed from the Menu to assign user-defined params
```

```
    hold = GameObject.FindWithTag("Holder").GetComponent<HoldValues>();
```

```
    mutationRate = hold.MR;
```

```
    popSize = hold.PS;
```

```
    gm = GameObject.FindWithTag("GameManager").GetComponent<GameManager>();
```

```
}
```

```
private void Start()
```

```
{
```

```
    initPop();
```

```
}
```

These lists hold info about the population; the NN at MatingPool[i] is calculated to have the fitness at Fitness[i]. These Lists are essentially corresponding lookup tables.

Technical B skill: Static members

```
private void initPop()
{
    // Spawn the required number of agents and add to Population List
    // So we can keep track of them
    for (int i = 0; i < popSize; i++) {
        Population.Add(Instantiate(agentPrefab));
    }
    Initialising = false;
}

public float Fitness(Transform agent, float jumpCount, float slideCount)
{
    float res = agent.position.z;
    if (jumpCount > 0 && slideCount > 0) {
        res -= ((1/jumpCount) + (1/slideCount));
    }
    return res;
}

private NN selectParent()
{
    // Algorithm adapted from D. Shiffman Genetic Algorithm video series
    // https://www.youtube.com/watch?v=9zfeTw-uFCw&list=PLRqwx-
    V7Uu6bJM3VgzjNV5YxVxUwzALHV&index=1
    // Start at 0
    int index = 0;

    // Pick a random number between 0 and 1
    float r = Random.Range(0f, 1f);

    // Keep subtracting probabilities until you get less than zero
    // Higher probabilities will be more likely to be selected since they will
    // Subtract a larger number towards zero
    while (r > 0) {
        r -= Fitnesses[index];
        // And move on to the next
        index++;
    }

    // Go back one to the fitness that was chosen
    index--;

    NN res = MatingPool[index];
```

Technical A algorithms:
Dynamic object
instantiation according to
PopSize and tracking
(Population List)

Excellent Coding Practice:
Modules with appropriate
interfaces

Fitness Function detailed in
Design Section.

**Technical A
algorithm:** Weighted
Random selection:
(Probabilistic
technique)

Returning the selected
NN; will be used as a
'parent' genome

```
        return res;
    }

    private void Update()
    {
        if (GenerationNum < 0) {
            GenerationNum = 0;
        }
    }
}
```

Good Coding Practice:
Defensive programming

```
private (NN, NN) crossover(NN parentA, NN parentB)
{
    NN childA = new NN();
    NN childB = new NN();
    childA.Init();
    childB.Init();
```

Technical A algorithms (follows to next page): crossover and mutation (genetic operators)

```
    // Perform an alternating crossover with parents A and B
    for (int i = 0; i < parentA.weights.Count; i++) {
        if (i % 2 == 0) {
            childA.weights[i] = parentA.weights[i];
        } else {
            childA.weights[i] = parentB.weights[i];
        }
    }

    for (int i = 0; i < parentA.weights.Count; i++) {
        if (i % 2 == 0) {
            childB.weights[i] = parentB.weights[i];
        } else {
            childB.weights[i] = parentA.weights[i];
        }
    }

    return (childA, childB);
}
```

Change from Design: Alternating crossover algorithm chosen instead of two-point crossover. It is simpler to implement and of similar effectiveness. It works by taking the two parents and creating two new children by alternating the weights of the parent's NNs.

Technical A Algorithms: Elementwise updating and handling of Matrix elements

Technical B Algorithms: Insertion Algorithm

```
private NN mutate(NN nn)
{
    for (int i = 0; i < nn.weights.Count; i++) {
        float prob = Random.Range(0f, 1f);
        if (prob < mutationRate) {
            int rowInd = Random.Range(0, nn.weights[i].RowCount);
            int colInd = Random.Range(0, nn.weights[i].ColumnCount);
```

Mutates a random weight in each layer with probability mutationRate.

```
        nn.weights[i][rowInd, colInd] = Random.Range(-1f, 1f);
    }
}

for (int i = 0; i < nn.biases.Count; i++) {
    float prob = Random.Range(0f, 1f);
    if (prob <= mutationRate) {
        nn.biases[i] = Random.Range(-1f, 1f);
    }
}

return nn;
}
```

Same thing is done for
biases

This method puts all of the GA
functions together to create
the new generation from the
previous one.

```
// Runs once all of this generation's agents are dead
public void EndGeneration()
```

Need every
fitness between
0 and 1 for
parent selection
algorithm to
work.

```
{
    nextGen.Clear();
    nextGen.Add(MatingPool[MatingPool.Count - 1]);
    int offset = 0;
    if (popSize % 2 == 0) {
        offset = 1;
        nextGen.Add(MatingPool[MatingPool.Count - 2]);
    }
    Fitnesses = normalizeFitnesses();
    for (int i = 0; i < Mathf.Floor(popSize / 2) - offset; i++) {
        NN parentA = selectParent();
        NN parentB = selectParent();
        (NN childA, NN childB) = crossover(parentA, parentB);
        nextGen.Add(mutate(childA));
        nextGen.Add(mutate(childB));
    }
    // Empty out all the volatile contents of the previous generation
    MatingPool.Clear();
    Fitnesses.Clear();
    Population.Clear();
    // Now we have begun the next generation
    GenerationNum++;
    Initialising = true;
}
```

Change from Requirements: Here I am implementing elitism. Recalling [Requirement 7a](#), I said that the top 2 agents of the previous generation would be preserved. In this implementation, only 1 is preserved if the population size is odd as this preserves a constant population size.

Default param that will be true if the 'Reset Evolution' button is pressed. This allows the method to init a new generation either for resetting purposes or simply because the algorithm is on the next generation.

```
public void InitNewGen(bool reset=false)
{
    for(int i = 0; i < popSize; i++) {
        GameObject newAgent = Instantiate(agentPrefab);
```

```
// Only update the Population with the next gen NNs if we are not resetting the evolution
// (Throws a NullReferenceException otherwise)
if(!reset) {
    newAgent.transform.Find("AgentNN").GetComponent<AgentNN>().nn = nextGen[i];
}
Population.Add(newAgent);
}
Initialising = false;
}
```

Called when an Agent collides with
Obstacle or veers too far to the side

```
public void AgentDeath(NN nn, float fitness, GameObject agent)
{
    MatingPool.Add(nn);
    Fitnesses.Add(fitness);
    Population.Remove(agent);
}
```

Saves the Agent's performance and the
NN that controlled it.

```
// De-spawns Agent from Scene
Destroy(agent);
}
```

Technical C Algorithm: Arithmetic mean

```
public float AverageFitness()
{
    return Fitnesses.Sum() / popSize;
}
```

```
private List<float> normalizeFitnesses()
{
    // Takes every fitness value in Fitnesses array
    // And puts them in the 0-1 range
    // Algorithm from: https://stats.stackexchange.com/questions/70801/how-to-normalize-data-to-0-1-range
    float minFitness = Fitnesses.Min();
    float maxFitness = Fitnesses.Max();
    float deltaF = maxFitness - minFitness;
    List<float> normalizedFitnesses = new List<float>();
    foreach (float fitness in Fitnesses) {
        normalizedFitnesses.Add((fitness - minFitness) / deltaF);
    }
    return normalizedFitnesses;
}
```

Technical B Algorithm: Array
Normalisation

```
private void clearGAParams()
{
}
```

```
GenerationNum = 0;
nextGen.Clear();
foreach (GameObject a in Population) {
    Destroy(a);
}
Population.Clear();
MatingPool.Clear();
Fitnesses.Clear();
}

public void ResetEvolution()
{
    clearGAParams();
    gm.Restart(true);
}

public void EndEvolution()
{
    clearGAParams();
    GameEnded = true;
    gm.QuitToMenu();
    Destroy(gameObject);
}
}
```

Called when ResetEvolution button pressed.

This method can be used in any script to Destroy the GameObject the script is attached to. A new GAManager GameObject will be instantiated if the user chooses to start the visualisation again.

PlayerMove.cs

This script governs everything to do with and Agent physically moving in the game world. Things like collisions with obstacles are handled in a separate script as those involve other Unity GameObjects. A lot of the code here is from the 2020 prototype and, as mentioned before, is overhead processing not relevant to the scope of this project. I will only annotate segments of code that were either added for this project or had to be modified to fit the needs of the project, or any concepts that are otherwise important .

```
using UnityEngine;

public class PlayerMove : MonoBehaviour
{
    private Rigidbody rb;
    private BoxCollider col;
    public LayerMask groundLayers;

    [Header("Move controls")]
}
```

This is a component that signifies that the GameObject should be controlled by Unity's own Physics engine.

```
public float forwardSpeed;
public float sideSpeed;
public float moveSpeed;
private float tempSpeed;

public float horizontal;

[Header("Speed Increase")]
public float speedMultiplier;
public float speedIncreaseMilestone;
public float maxSpeed;

private float speedMilestoneCount;

public bool isJumpPressed = false;
private float forwardSpeedWhileInAir;

[Header("Jump controls")]
public float jumpSpeed;
public float fallMultiplier;
public float lowJumpMultiplier;

public bool isSliding = false;
public bool slideStopped = true;

private Vector3 moveHorizontal;
private Vector3 moveForward;
private Vector3 movement;

private Animator anim;

void Start()
{
    forwardSpeedWhileInAir = forwardSpeed;
    tempSpeed = forwardSpeed;

    rb = GetComponent<Rigidbody>();
    col = GetComponent<BoxCollider>();

    anim = GetComponent<Animator>();
}
```

This had to be added to ensure that jumps were more realistic.

Good Programming Practice: Self-Documenting Code

Basic Programming Practice: Meaningful identifier names

```
void Update()
```



```
{
    moveHorizontal = new Vector3(horizontal, 0 , 0) * sideSpeed;
    moveForward = new Vector3(0, 0 , forwardSpeed);
    movement = moveHorizontal + moveForward;

    if (rb.velocity.y < 0) {
        rb.velocity += Vector3.up * Physics.gravity.y * (fallMultiplier - 1) * Time.deltaTime;
    } else if (rb.velocity.y > 0 && !isJumpPressed) {
        rb.velocity += Vector3.up * Physics.gravity.y * (lowJumpMultiplier - 1) * Time.deltaTime;
    }

    if (IsGrounded()) {
        if (isSliding) {
            if (!isJumpPressed) {
                anim.SetBool("Slide", true);
                anim.SetBool("Run", false);
            } else {
                anim.SetBool("Slide", false);
            }
            Slide();
        } else if (slideStopped) {
            resetSlide();
        }
    }

    if (transform.position.z > speedMilestoneCount && forwardSpeed < maxSpeed && IsGrounded()) {
        speedMilestoneCount += speedIncreaseMilestone;
        speedIncreaseMilestone *= speedMultiplier;

        forwardSpeed *= speedMultiplier;
        tempSpeed = forwardSpeed;
    }
}

void FixedUpdate()
{
    if (IsGrounded()) {
        forwardSpeed = tempSpeed;
        tempSpeed = forwardSpeed;
    } else {
        forwardSpeed = forwardSpeedWhileInAir;
    }
    if (IsGrounded() && !isSliding) {
```

MonoBehaviour Method: Unlike Update(), FixedUpdate() is framerate-independent; it can be called as many times as is needed per frame. It is called to handle Physics calculations.

```
        anim.SetBool("Run", true);
        anim.SetBool("Jump", false);
        anim.SetBool("Fall", false);
        anim.SetBool("Slide", false);
        anim.SetBool("Flip", false);
    } else if (IsGrounded() && isSliding) {
        anim.SetBool("Run", false);
        anim.SetBool("Jump", false);
        anim.SetBool("Fall", false);
        anim.SetBool("Slide", true);
        anim.SetBool("Flip", false);
    }

    // Handle Jumping
    if (IsGrounded() && isJumpPressed) {
        if (isSliding) {
            anim.SetBool("Run", false);
            anim.SetBool("Slide", false);
            anim.SetBool("Fall", false);
            anim.SetBool("Flip", true);
        } else {
            anim.SetBool("Run", false);
            anim.SetBool("Slide", false);
            anim.SetBool("Jump", true);
        }

        rb.velocity = Vector3.up * jumpSpeed;

        FindObjectOfType<AudioManager>().Play("Jump");

    } else if (!IsGrounded() && !anim.GetBool("Jump")) {

        anim.SetBool("Run", false);
        anim.SetBool("Slide", false);
        anim.SetBool("Fall", true);
    }

    moveCharacter(movement);
}

void moveCharacter(Vector3 desired)
{
    rb.MovePosition(transform.position + (desired * moveSpeed * Time.fixedDeltaTime));
}
```

```
private bool IsGrounded()  
{  
    return Physics.CheckCapsule(col.bounds.center, new Vector3(col.bounds.center.x,  
col.bounds.min.y, col.bounds.center.z), col.size.x * .9f, groundLayers);  
}  
  
void Slide()  
{  
    col.center = new Vector3(0f, 0.4f, 0f);  
    col.size = new Vector3(1f, 0.8f, 1f);  
}  
  
void resetSlide()  
{  
    col.center = new Vector3(0f, 0.9f, 0);  
    col.size = new Vector3(1f, 1.8f, 1f);  
    anim.SetBool("Slide", false);  
    anim.SetBool("Run", true);  
}  
}
```

PlayerCollision.cs

As the name implies, this script contains methods that handle the logic of when an agent collides with another object in the scene. Depending on what the collided object was, different things need to happen; the agent should either die or ignore the collision, and the appropriate animations need to play.

```
using UnityEngine;  
  
public class PlayerCollision : MonoBehaviour  
{  
    public PlayerMove movement;  
    private Animator anim;  
    private Rigidbody rb;  
    private float currFitness;  
    public AgentNN agentNN;  
    GAManager gm;  
    private bool dead = false;  
  
    private void Awake()
```

```
{
    anim = GetComponent<Animator>();
    rb = GetComponent<Rigidbody>();
    gm = FindObjectOfType<GAManager>();
    audioManager = FindObjectOfType<AudioManager>();
}

private void Update()
{
    if ((transform.position.y < -1) || (Mathf.Abs(transform.position.x) > 7)) {
        dead = true;
    }

    if (dead) {
        currFitness = gm.Fitness(transform, agentNN.JumpCount, agentNN.SlideCount);
        gm.AgentDeath(agentNN.nn, currFitness, gameObject);
    }
}

private void OnCollisionEnter (Collision collided)
{
    if (collided.collider.tag == "Spike" ||
        collided.collider.tag == "RedSpikes" ||
        collided.collider.tag == "Boombox" ||
        collided.collider.tag == "Enemy") {

        if (collided.collider.tag == "Boombox" && movement.isSliding) {
            Physics.IgnoreCollision(collided.gameObject.GetComponent<Collider>(),
GetComponent<Collider>());
        } else {
            anim.SetBool("Alive", false);
            anim.SetBool("Fall", false);
            anim.SetBool("Run", false);
            anim.SetBool("Jump", false);
            anim.SetBool("Slide", false);
            anim.SetBool("Flip", false);
            movement.enabled = false;

            dead = true;
        }
    }

    } else if (collided.collider.tag == "Agent") {
        // Agents must not collide with other agents
    }
}
```

Excellent Coding Practice: Loosely coupled modules

MonoBehaviour Method: This method is called when the GameObject's Collider component detects a collision.

```
        Physics.IgnoreCollision(collided.gameObject.GetComponent<Collider>(),  
GetComponent<Collider>());  
    }  
  
}  
}
```

NN.cs

This script actually does not inherit from MonoBehaviour, as it alone does not need to execute anything in the Unity. NN.cs is actually a class built mostly in vanilla C#. In fact, the only reason that the UnityEngine namespace is also included in this script is so that Unity's built in Random function can be used rather than the vanilla C# one. This is done for consistency, as all other mathematical calculations are done using Unity's Mathf library wherever possible.

```
using System;  
using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using MathNet.Numerics.LinearAlgebra;  
using Random = UnityEngine.Random;
```

```
public class NN
```

```
{
```

```
    // NN Hyperparameters
```

```
    public static int numHiddenNodes = 8;
```

```
    public static int numHiddenLayers = 3;
```

```
    public static int numInputs = 10;
```

```
    public static int numOutputs = 5;
```

```
    // 1 x 10 matrix as there are 10 inputs
```

```
    public Matrix<float> inputLayer = Matrix<float>.Build.Dense(1, numInputs);
```

```
    // We can modify the number of hidden layers if we want so we need a list of matrices
```

```
    public List<Matrix<float>> hiddenLayers = new List<Matrix<float>>();
```

```
    // 5 outputs
```

```
    public Matrix<float> outputLayer = Matrix<float>.Build.Dense(1, numOutputs);
```

```
    // Weights between layers
```

```
    public List<Matrix<float>> weights = new List<Matrix<float>>();
```

Technical A Skill: Complex mathematical model (Neural Network)

Technical B Skill: Simple OOP model

Technical B Algorithm: Static members

Technical B skill: Multi-dimensional collections of iterables.

```
// List of biases which corresponds to the bias of a layer
public List<float> biases = new List<float>();

public void Init()
{
    // Clear all matrices of garbage values
    inputLayer.Clear();
    hiddenLayers.Clear();
    outputLayer.Clear();
    weights.Clear();
    biases.Clear();

    // Populate hidden layers
    PopulateNN();
    RandomiseWeights();
}

public void PopulateNN()
{
    // Handle hidden layers and weights
    for (int i = 0; i < numHiddenLayers; i++) {
        Matrix<float> newHiddenLayer = Matrix<float>.Build.Dense(1, numHiddenNodes);
        hiddenLayers.Add(newHiddenLayer);
        biases.Add(Random.Range(-1f, 1f));

        // Matrix for input to first hidden layer will have different dimensions, handle separately
        if (i == 0) {
            Matrix<float> firstWeights = Matrix<float>.Build.Dense(numInputs, numHiddenNodes);
            weights.Add(firstWeights);
        }

        if (i < numHiddenLayers - 1) {
            Matrix<float> newWeights = Matrix<float>.Build.Dense(numHiddenNodes, numHiddenNodes);
            weights.Add(newWeights);
        }
    }

    // Matrix for last hidden to output layer
    Matrix<float> outputWeights = Matrix<float>.Build.Dense(numHiddenNodes, numOutputs);
    weights.Add(outputWeights);
    biases.Add(Random.Range(-1f, 1f));
}

public void FeedForward()
{
```



Throughout this method, **Technical A algorithm**: Matrix multiplication (advanced matrix operations).

```
// Activate Input Layer
inputLayer = inputLayer.PointwiseTanh();

// Input to first hidden layer
hiddenLayers[0] = (inputLayer * weights[0]) + biases[0];
hiddenLayers[0].PointwiseTanh();

// First to penultimate hidden layer
for (int i = 1; i < hiddenLayers.Count; i++) {
    hiddenLayers[i] = ((hiddenLayers[i-1] * weights[i]) + biases[i]).PointwiseTanh();
}

// Last hidden layer to output layer
outputLayer = ((hiddenLayers[hiddenLayers.Count-1] * weights[weights.Count-1]) +
biases[biases.Count-1]);

for (int i = 0; i < outputLayer.ColumnCount; i++) {
    // Sigmoid every element in output vector
    outputLayer[0, i] = sigmoid(outputLayer[0, i]);
}

}

public void RandomiseWeights()
{
    for (int i = 0; i < weights.Count; i++) {
        for (int j = 0; j < weights[i].RowCount; j++) {
            for (int k = 0; k < weights[i].ColumnCount; k++) {
                weights[i][j, k] = Random.Range(-1f, 1f);
            }
        }
    }
}

private float sigmoid(float z)
{
    return (1 / (1 + Mathf.Exp(-z)));
}

}
```

Technical A algorithm: Use of hyperbolic tangent function

Technical A algorithm: Implementation of Sigmoid activation function.

Excellent Coding Practice: Cohesive modules.

SensorSystem.cs

This is the script that handles the SensorSystem I designed. It instantiates them, detects any collisions with Raycasting, and colours the sensors appropriately (you can only see the sensors in the Editor View with the 'Gizmos' button toggled; the sensors will not be visible in the final build of the game). Additionally, the script also has a method that will 'pass on'

the Sensor info to the Agent's NN. This can be likened to a human detecting something with their eyes, and then having their eyes send that information to the brain to be processed.

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using MathNet.Numerics.LinearAlgebra;

public class SensorSystem : MonoBehaviour
{
    [SerializeField]
    // List of obstacle objects (Spikes, Boomboxes, Enemies etc.)
    private GameObject[] obstacles;

    // Dictionary mapping tags of each obstacle type to a numerical ID e.g. "Spike" -> 0
    private Dictionary<string, int> tagsToNum = new Dictionary<string, int>();

    // Reference to the agent's AgentNN script
    public AgentNN agentNN;

    // Agent's maximum sense distance
    public float maxViewDistance = 100f;

    int cols;

    private void Start()
    {
        // Populating tagsToNum dictionary by mapping obstacle type's tag to a unique numerical ID
        int val = 0;
        foreach (GameObject obstacle in obstacles) {
            tagsToNum[obstacle.tag] = val;
            val++;
        }

        cols = agentNN.nn.inputLayer.ColumnCount;
    }

    private void FixedUpdate()
    {
        handleSensors();
    }
}
```

Technical A skill: SensorSystem data structure (complex scientific model)

Technical B skill: Dictionaries

Technical B algorithm: Simple and appropriate updating algorithms of dictionary values


```
private Vector3[] instantiateDirVectors()
{
    // Returns a list of 5 vectors at increasing angles around center of the agent's transform
    Vector3 left = -transform.right;
    Vector3 fLeft = -transform.right + 0.5f*transform.forward;
    Vector3 forward = transform.forward;
    Vector3 fRight = transform.forward + 0.5f*transform.right;
    Vector3 right = transform.right;
    Vector3[] dirVectors = {left, fLeft, forward, fRight, right};
    return dirVectors;
}

private void handleSensors()
{
    Vector3[] dirVectors = instantiateDirVectors();

    // Arrays whose corresponding elements we will pair up and pass into the Agent's NN
    List<float> hitDistances = new List<float>();
    List<int> collidedObjects = new List<int>();

    // This ray is what we use to check for collisions
    Ray ray = new Ray(transform.position, dirVectors[0]);

    // The RaycastHit object is what we use to store the collision event
    RaycastHit hit;

    float hitDistanceToAdd;
    int collisionTagToAdd;

    // Check collision for each sensor
    foreach(Vector3 v in dirVectors) {

        // Set the ray object to be in the direction of one of our sensors
        ray.direction = v;

        // Detecting any collision (provided it is within the agent's view distance)
        if (Physics.Raycast(ray, out hit) && hit.distance <= maxViewDistance)
        {
            if (hit.collider.gameObject.tag != "Agent" && hit.collider.gameObject.tag !=
"Untagged") {

                // Color invisible wall sensors blue
                if (hit.collider.gameObject.tag == "LWall" || hit.collider.gameObject.tag == "
RWall") {
```

Technical C Skill: Single-dimensional arrays

The RaycastHit object is what we use to store info on a Ray collision event

Technical A algorithm: Raycast Vector calculations

```
        Debug.DrawLine(ray.origin, hit.point, Color.blue);
    } else {
        Debug.DrawLine(ray.origin, hit.point, Color.green);
    }

    // Add distance from the obstacle to the sensor's source to the distance array
    hitDistanceToAdd = hit.distance;
    // And also add the numerical ID of the obstacle type to this array
    collisionTagToAdd = tagsToNum[hit.collider.gameObject.tag];

} else {

    // Not detecting an obstacle means we color the sensor red
    Debug.DrawLine(ray.origin, hit.point, Color.red);

    // Add dummy values to list if obstacle not detected
    hitDistanceToAdd = -1f;
    collisionTagToAdd = -1;
}

} else {
    // Again, add dummy values if there was no RayCast event (no collision)
    hitDistanceToAdd = -1f;
    collisionTagToAdd = -1;
}

hitDistances.Add(hitDistanceToAdd);
collidedObjects.Add(collisionTagToAdd);
}

// Now populate agent's NN input layer with sensor info
linkToInputLayer(hitDistances, collidedObjects);
}

// This method sets the input layer of the agent's NN to the values of the sensor system
private void linkToInputLayer(List<float> hits, List<int> collisions)
{
    // Matrix which we will replace the current input layer of the NN with
    Matrix<float> newInputs = Matrix<float>.Build.Dense(1, cols);

    int hitInd = 0;
    int colInd = 0;
    int ind1 = 0;
    int ind2 = 1;

    // Populate with sensor vals
```

Debug commands
aren't visible in a
compiled build.

Technical A algorithm:
Dynamically passing
data from raycast
sensors to neural
network inputs.

```
// Iterating through both the hits and collisions array simultaneously
for (int i = 0; i < cols; i++) {
    // Inputs at even indexes in the input layer are for the hit distances
    if (i % 2 == 0) {
        newInputs[0, ind1] = hits[hitInd];
        ind1 += 2;
        hitInd++;
    } // Inputs at odd indexes in the input layer are for collision tags
    else {
        newInputs[0, ind2] = (float)collisions[colInd];
        ind2 += 2;
        colInd++;
    }
}

// Now we replace the current input layer with this new one we have populated
agentNN.nn.inputLayer = newInputs;
}
```

Technical A Algorithms:
Elementwise updating
and handling of Matrix
elements

Technical B algorithm:
Insertion Algorithm

AgentNN.cs

This script acts as a sort of relation between the NN class and a given Agent's PlayerMove script (which recall, is a component attached to this specific Agent GameObject). After SensorSystem.cs passes its information as inputs to the NN, this script will then take that NN, feed it forward, then prompt the agent to take the appropriate action based on what index the output node with the maximum value was. In other words, whatever the agent 'thinks' is the best move to take, this script allows the agent to do so.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using MathNet.Numerics.LinearAlgebra;
```

```
// This script represents an instance of a relationship between ONE NN and ONE Agent
```

```
public class AgentNN : MonoBehaviour
{
```

```
    public NN nn;
    public Transform agent;
    public PlayerMove movement;
```

Good Coding Practice:
Modularisation of Code

```
    // Keeping track of jump and slide counts to use in fitness calculation
    public int JumpCount
    { get; set; }
```

```
public int SlideCount
{ get; set; }

// These are two ensure that the jumpCount doesn't get incremented too many times per frame
float jumpCoolDown = 10f;
float jumpCoolDownTimer;

void Awake()
{
    nn = new NN();
    nn.Init();
}

void Update()
{
    if (jumpCoolDownTimer > 0) {
        jumpCoolDownTimer -= Time.deltaTime;
    } else if (jumpCoolDownTimer < 0) {
        jumpCoolDownTimer = 0f;
    }

    nn.FeedForward();
    float maxOutput = 0f;
    int maxOutputInd = 0;

    // Find the index of the output node with the maximum value
    // (i.e. the move that the NN is most confident to take)
    for (int i = 0; i < nn.outputLayer.ColumnCount; i++) {
        if (nn.outputLayer[0, i] > maxOutput) {
            maxOutput = nn.outputLayer[0, i];
            maxOutputInd = i;
        }
    }

    // Choose max output and perform the move corresponding to that output node
    switch(maxOutputInd) {
        // LEFT (A key)
        case 0:
            movement.horizontal = -1f;
            movement.isJumpPressed = false;
            movement.isSliding = false;
            movement.slideStopped = true;
            break;
    }
}
```

Technical B Algorithm:
Generation of objects based
on simple OOP model

Mappings
between output
nodes and moves
are arbitrary

Technical C Skill: Single-dimensional array

Technical C Algorithm: Linear search for
max element

Good Coding Practice: Switch
cases

```
// RIGHT (D key)
case 1:
    movement.horizontal = 1f;
    movement.isJumpPressed = false;
    movement.isSliding = false;
    movement.slideStopped = true;
    break;

// DO NOTHING
case 2:
    movement.horizontal = 0f;
    movement.isJumpPressed = false;
    movement.isSliding = false;
    movement.slideStopped = true;
    break;

// JUMP (Spacebar)
case 3:
    if (jumpCoolDownTimer == 0) {
        movement.horizontal = 0f;
        movement.isJumpPressed = true;
        movement.isSliding = false;
        movement.slideStopped = true;

        jumpCoolDownTimer = jumpCoolDown;

        JumpCount++;
    }
    break;

// SLIDE (S key)
case 4:
    movement.horizontal = 0f;
    movement.isSliding = true;
    movement.isJumpPressed = false;

    SlideCount++;
    break;
}
}
}
```

CourseManager.cs

This script handles how the course is generated as the agents are running. It would not be a wise design decision to simply have the sections of the course continually spawn in front and not worry about the sections behind us. Eventually, we would run out of memory. This script is meant for handling when to set sections of the course ('Tiles') as active, as well as which Tiles to spawn.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CourseManager : MonoBehaviour
{
    public Transform cam;
    public GameObject[] prefabs;
    private List<GameObject> activeTiles = new List<GameObject>();
    public float zSpawn = 0f;
    public float tileLength;
    public int numFirstTiles = 5;
    void Start()
    {
        SpawnFirstTiles();
    }

    public void SpawnFirstTiles()
    {
        zSpawn = 0f;
        ClearActiveTiles();
        for (int i = 0; i < numFirstTiles; i++) {
            if (i == 0) {
                SpawnTile(0);
            } else {
                SpawnTile(Random.Range(0, prefabs.Length));
            }
        }
    }

    void Update()
    {
        // Checks if player has reached the end of the tile they are on
        if (cam.position.z - 50 > zSpawn - (numTiles * tileLength)) {
            // Spawn random new tile from prefabs list
            SpawnTile(Random.Range(0, prefabs.Length));
            DeleteOldestTile();
        }
    }
}
```

Throughout script, **Good Coding Practice:** Self-documenting code

```
    }  
}  
  
void SpawnTile(int tileIndex)  
{  
    // Instantiate new tile with correct orientation  
    GameObject currentTile = Instantiate(prefabs[tileIndex], transform.forward * zSpawn,  
Rotate(transform.rotation));  
    activeTiles.Add(currentTile);  
    // Next tile will spawn one tile length away  
    zSpawn += tileLength;  
}  
  
void ClearActiveTiles()  
{  
    foreach (GameObject tile in activeTiles) {  
        Destroy(tile);  
    }  
    activeTiles.Clear();  
}  
  
// This function makes sure the tiles are facing the right way  
Quaternion Rotate(Quaternion baseRot)  
{  
    Quaternion rot = baseRot;  
    rot = Quaternion.Euler(0 , 90, 0);  
    return rot;  
}  
  
void DeleteOldestTile()  
{  
    Destroy(activeTiles[0]);  
    activeTiles.RemoveAt(0);  
}  
}
```

Technical A algorithms: Dynamic
object instantiation

Excellent Coding Practice: Cohesive
modules

FollowPlayer.cs

The camera in a Unity scene is another GameObject like anything else in the scene. As such, it has a Transform that determines its position. This script ensures that all agents will be visible by the camera (and hence, the user) at any given point. It does this by travelling at the speed of the slowest agent.

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College.
Centre Number: 74615. Component Code: 7517/C.

```
using System.Linq;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FollowPlayer : MonoBehaviour
{
    public float moveSpeed;
    float slowestAgentSpeed;
    GameObject[] agents;
    Vector3 startPos;

    void Start()
    {
        startPos = transform.position;
    }

    void Update()
    {
        agents = GameObject.FindGameObjectsWithTag("Agent");
        if (agents.Count() > 0) {
            slowestAgentSpeed = agents.Min(agents => agents.GetComponent<PlayerMove>().forwardSpeed);
            transform.Translate(new Vector3(0f, 0f, slowestAgentSpeed * moveSpeed * Time.deltaTime),
Space.World);
        }
    }

    public void ResetPos()
    {
        transform.position = startPos;
    }
}
```

WindowGraph.cs

This is the script that controls the logic of the graph of average fitness over time over the course of a given training session. The graph is positioned at the bottom left corner of the screen. In my [Design](#) pg. 54, I stated that I would be getting the graph functionality from a third-party Asset called 'Dynamic Line Chart'. Unfortunately, this package has, as of the writing of this project, been pulled from the Asset Store, and so its codebase is no longer available. Luckily, I managed to find a YouTube tutorial by the user CodeMonkey* for making an even better line graph. The code here was inspired by that tutorial, albeit with modifications and extensions to suit my project.


```
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using UnityEngine.UI;
public class WindowGraph : MonoBehaviour
{
    // Graph code adapted from https://www.youtube.com/watch?v=CmU5-v-v1Qo
    [SerializeField]
    private Sprite circleSprite;
    private RectTransform graphContainer;

    // Lists of the objects corresponding to the points and lines seen in the graph window
    private List<GameObject> points;
    private List<GameObject> lines;

    // List of average fitnesses that the graph will iterate over and display the values
    private List<float> valueList;

    // Maximum number of points that can be displayed on the graph at any time
    private int maxValues = 50;

    private void Awake()
    {
        graphContainer = transform.Find("GraphContainer").GetComponent<RectTransform>();
        points = new List<GameObject>();
        lines = new List<GameObject>();
        valueList = new List<float>();
    }

    private GameObject createCircle(Vector2 anchoredPosition)
    {
        // Instantiating a new circle object and setting its position parameters to the function
parameter
        GameObject circleObject = new GameObject("circle", typeof(Image));
        circleObject.transform.SetParent(graphContainer, false);
        circleObject.GetComponent<Image>().sprite = circleSprite;
        RectTransform rectTransform = circleObject.GetComponent<RectTransform>();
        rectTransform.anchoredPosition = anchoredPosition;
        rectTransform.sizeDelta = new Vector2(11, 11);
        rectTransform.anchorMin = new Vector2(0, 0);
        rectTransform.anchorMax = new Vector2(0, 0);
        return circleObject;
    }
}
```

Good Coding Practice: Well-designed user interface

A RectTransform is a component used to position and orient UI elements.

```
public void ShowGraph()
{
    // In the unexpected case that there are no values to show, break out of the function
    if (valueList.Count <= 0) {
        return;
    }

    // Make graph 'responsive' to extreme y values
    float buffer = 0.2f;

    float graphHeight = graphContainer.sizeDelta.y;
    float graphWidth = graphContainer.sizeDelta.x;

    float yMax = valueList.Max();
    float yMin = valueList.Min();
    float yDiff = Mathf.Max(yMax - yMin, 5f);
    yMax += (yDiff * buffer);
    yMin -= (yDiff * buffer);
    float xSize = graphWidth / (valueList.Count + 1);

    GameObject prevCircleGameObject = null;
    GameObject circleGameObject = null;

    for (int i = 0; i < valueList.Count; i++) {
        float xPos = xSize + i * xSize;
        float yPos = ((valueList[i] - yMin) / (yMax - yMin)) * graphHeight;

        if (points.Count > 0) {
            circleGameObject = points[points.Count - 1];
        }

        points.Add(createCircle(new Vector2(xPos, yPos)));

        // Iteratively adding connections between neighboring points
        if (prevCircleGameObject != null) {
            createDotConnection(prevCircleGameObject.GetComponent<RectTransform>().anchoredPosition,
            circleGameObject.GetComponent<RectTransform>().anchoredPosition);
        }
        prevCircleGameObject = circleGameObject;
    }

    // Need to account for the last point pair separately
```

```
        createDotConnection(points[points.Count - 1].GetComponent<RectTransform>().anchoredPosition,
points[Mathf.Max(0, points.Count - 2)].GetComponent<RectTransform>().anchoredPosition);
    }

    private void createDotConnection(Vector2 dotPosA, Vector2 dotPosB)
    {
        // Create a line game object to connect two points on the graph
        GameObject lineObject = new GameObject("dotConnection", typeof(Image));
        lineObject.transform.SetParent(graphContainer, false);
        lineObject.GetComponent<Image>().color = new Color(1, 1, 1, 0.5f);
        RectTransform rectTransform = lineObject.GetComponent<RectTransform>();
        Vector2 dir = (dotPosB - dotPosA).normalized;
        float distance = Vector2.Distance(dotPosA, dotPosB);
        rectTransform.anchorMin = new Vector2(0, 0);
        rectTransform.anchorMax = new Vector2(0, 0);
        rectTransform.sizeDelta = new Vector2(distance, 10f);
        rectTransform.anchoredPosition = dotPosA + dir * distance * 0.5f;
        rectTransform.localEulerAngles = new Vector3(0,0, (dir.y < 0 ? -Mathf.Acos(dir.x) :
Mathf.Acos(dir.x)) * Mathf.Rad2Deg);
        lines.Add(lineObject);
    }

    // Clear the graph window for the next generation
    public void ClearGraph(bool resetEvolution=false)
    {
        foreach (GameObject point in points) {
            Destroy(point);
        }
        foreach (GameObject line in lines) {
            Destroy(line);
        }

        points.Clear();
        lines.Clear();

        // If this method is instead called when the user resets the evolution, the whole valueList is
        // Cleared, giving us a fresh graph
        if (resetEvolution == true) {
            valueList.Clear();
        }
    }

    public void AddNewPoint(float val)
    {

```

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College.
Centre Number: 74615. Component Code: 7517/C.

```
// Simply recording the new point information in valueList to be displayed in the next
// ShowGraph() call
valueList.Add(val);

// Only display the most recent points
if (valueList.Count > maxValues) {
    valueList.RemoveRange(0, Mathf.Min(valueList.Count - maxValues, valueList.Count));
}
}
}
```

StatManager.cs

This script is for the statistics that are shown on the screen at any given time (at the top left of the screen). The statistics shown are: Population Size, Mutation Rate, current generation number, number of alive agents remaining in the current generation, and the max fitness so far in the current generation. Some of these need to make use of the Holder GameObject's [HoldValues.cs](#) component.

```
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using UnityEngine.UI;

public class StatManager : MonoBehaviour
{
    public Text statText;
    HoldValues hold;
    GAManager genetic;
    void Start()
    {
        hold = GameObject.Find("Holder").GetComponent<HoldValues>();
        genetic = GameObject.Find("GAManager").GetComponent<GAManager>();
    }

    void Update()
    {
        statText.text = $"Pop Size: {hold.PS}\nMutation Rate: {hold.MR}\n\nCurrent Gen:
{genetic.GenerationNum}\nAgents Remaining: {GameObject.FindGameObjectsWithTag("Agent").Length}\nMax
Fitness: {(genetic.Fitnesses.Count > 0 ? genetic.Fitnesses.Max(): 0f)}";
    }
}
```

MainMenu.cs

This is the main script for handling logic in the StartMenu scene, which the user must progress through in order to start the visualisation. Regardless, defensive programming techniques were enplaced in [GameManager.cs](#) to ensure that the program would not crash if this did not occur (i.e. the player was somehow able to load the MainScene without having navigated the StartMenu).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour
{
    public GameObject options;
    public GameObject paramInput;

    public void Start()
    {
        gameObject.SetActive(true);
        options.SetActive(false);
        paramInput.SetActive(false);
    }

    // Executed when Start Button clicked
    public void StartVisualisation()
    {
        // Loads next Scene in the queue (this is the main visualisation scene as there are only two
        scenes)
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

    // Executed when Quit Button Clicked
    public void Quit()
    {
        // Application only quits in an actual build of the project, NOT in the Unity Editor, so this
        debug log is to verify the function executes on quit button click
        Debug.Log("QUIT");
        Application.Quit();
    }
}
```

OptionsMenu.cs

This menu script handles the logic of the Options Menu, which can be accessed via the Main Menu. The only option to control is the music, which is a **change from my Design section**. In my [Design](#), pg. 52, I said that there would be sound effects for jumping, dying etc. and that the user could control the volume of these effects in the options menu. However, after repeated runs when debugging and testing my system, the sound effects quickly become irritating and detract from the functionality from the system, especially when there are a very large number of agents. Thus, I have removed the sound effects, and hence the option to control their volume; the only audio in the project is music.

```
using System.Collections;
using System.Collections.Generic;
using System;
using UnityEngine;
using UnityEngine.UI;

public class OptionsMenu : MonoBehaviour
{
    public Slider musicSlider;
    public AudioManager am;
    private Sound themeMusic;
    void Start()
    {
        themeMusic = Array.Find(am.sounds, sound => sound.name == am.musicName);
        themeMusic.volume = musicSlider.value;
    }

    // Executed on change of music slider
    public void UpdateMusicVolume()
    {
        // Set value of music volume to music slider value
        am.UpdateVolume(themeMusic, musicSlider.value);
    }
}
```

GAParamInput.cs

The final menu in the initial scene is the one that deals with user input of population size and mutation rate. This script is attached to that UI GameObject. It relies on a record object that uses the [HoldValues.cs](#) component to store values between scenes. The inputted GA Params are used to populate the respective fields in the record, then the

MainScene is started by calling `StartVisualisation()` of MainMenu.cs when the 'Go' button is pressed.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
public class GAParamInput : MonoBehaviour
{
    string mutationRate = "";
    string popSize = "";
    public Text mrText;
    public Text psText;
    public GameObject mainMenu;

    [HideInInspector]
    public int PS
    { get; set; }

    [HideInInspector]
    public float MR
    { get; set; }

    private HoldValues holder;

    void Start()
    {
        holder = GameObject.Find("Holder").GetComponent<HoldValues>();
    }

    public void StoreData()
    {
        mutationRate = mrText.text;
        popSize = psText.text;

        if (!ValidateData(mutationRate, popSize)) {
            SceneManager.LoadScene(SceneManager.GetActiveScene().name);
        }
        else {
            PS = int.Parse(popSize);
            MR = float.Parse(mutationRate);
            holder.HoldGAParams();
            mainMenu.GetComponent<MainMenu>().StartVisualisation();
        }
    }
}
```

Good Coding Practice: Basic
Validation Methods

Good Coding Practice: Managed
casting of types

```
    }  
}  
  
bool ValidateData(string mr, string ps)  
{  
    float mrFlt;  
    int psInt;  
    try {  
        mrFlt = float.Parse(mr);  
        psInt = int.Parse(ps);  
    }  
    catch {  
        return false;  
    }  
  
    if (mrFlt < 0f || mrFlt > 1f || psInt < 2 || psInt > 50) {  
        return false;  
    }  
    return true;  
}  
}
```

Good Coding Practice: Managed casting of types

Excellent Coding Practice: Good exception handling

Basic Coding Practice: Basic Validation Methods

Change from Requirements: Population size maximum restricted to 50

AudioManager.cs

```
using System;  
using UnityEngine.Audio;  
using UnityEngine;  
  
public class AudioManager : MonoBehaviour  
{  
    public string musicName;  
    public Sound[] sounds;  
    public static AudioManager instance;  
  
    private void Awake()  
    {  
        if (instance == null) {  
            instance = this;  
        } else {  
            Destroy(gameObject);  
            return;  
        }  
    }  
}
```

Composition. Technical B Skill: Simple OOP Model


```
    }

    DontDestroyOnLoad(gameObject);
    foreach (Sound s in sounds) {
        s.source = gameObject.AddComponent<AudioSource>();
        s.source.clip = s.clip;

        s.source.volume = s.volume;
        s.source.pitch = s.pitch;
        s.source.loop = s.loop;
    }
}

private void Start()
{
    Play(musicName);
}

public void Play(string name)
{
    Sound s = Array.Find(sounds, sound => sound.name == name);
    if (s == null) {
        return;
    }
    s.source.Play();
}

public void UpdateVolume(Sound s, float volume)
{
    // Change Audio Manager volume value for this sound
    s.volume = volume;

    // Update actual sound volume from the sound source in the game.
    s.source.volume = volume;
}
}
```

Sound.cs

```
using UnityEngine.Audio;
using UnityEngine;

[System.Serializable]
```

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College.
Centre Number: 74615. Component Code: 7517/C.

```
public class Sound
{
    public string name;

    public AudioClip clip;

    [Range(0f, 1f)]
    public float volume;
    [Range(.1f, 3f)]
    public float pitch;

    public bool loop;

    [HideInInspector]
    public AudioSource source;
}
```

Technical B Skill: Simple OOP Model

Technical B Skill: Records

SpawnEnemies.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SpawnEnemies : MonoBehaviour
{
    public GameObject plane;
    public GameObject enemy;

    public float LOffset = 5f;
    public float ROffset = 5f;
    public float heightOffset = 0.2f;

    // Start is called before the first frame update
    void Awake()
    {
        float probL = Random.value;
        float probR = Random.value;

        if (probL >= 0.5) // 50% chance
        {
            GameObject enemyL = Instantiate(enemy, Left(transform.position),
                                              enemy.transform.rotation);
        }
    }
}
```

```
        enemyL.transform.parent = gameObject.transform;
    }

    if (probR >= 0.5)
    {
        GameObject enemyR = Instantiate(enemy, Right(transform.position),
                                         enemy.transform.rotation);

        enemyR.transform.parent = gameObject.transform;
    }
}

Vector3 Left(Vector3 pos)
{
    Vector3 temp = pos;
    temp.x -= LOffset;
    temp.y -= heightOffset;
    return temp;
}

Vector3 Right(Vector3 pos)
{
    Vector3 temp = pos;
    temp.x += ROffset;
    temp.y -= heightOffset;
    return temp;
}
}
```

HoldValues.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class HoldValues : MonoBehaviour
{
    GParamInput input;
    [HideInInspector]
    public int PS
    { get; set; }
    [HideInInspector]
```

This script acts like a record data structure; storing the state of variables between scenes that need to be preserved. These are mainly to do with the GA.

Technical B Skill: Records

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College.
Centre Number: 74615. Component Code: 7517/C.

```
public float MR
{ get; set; }

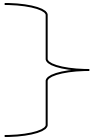
[HideInInspector]
public bool firstGen = true;

public int GenerationNum
{ get; set; }

public static HoldValues instance;
void Awake()
{
    if (instance == null)
        instance = this;
    else
    {
        Destroy(gameObject);
        return;
    }
    GenerationNum = 1;
    DontDestroyOnLoad(gameObject);
}

public void HoldGAParams()
{
    if (input != null)
    {
        PS = input.PS;
        MR = input.MR;
    }
}

void Update()
{
    // If we are on the menu screen (Scene 0) and the Param Input screen is currently up then
    // initialize input to the reference of the param input script
    if (SceneManager.GetActiveScene().buildIndex == 0 && GameObject.Find("GAParamInput") != null)
    {
        input = GameObject.Find("GAParamInput").GetComponent<GAParamInput>();
    }
}
}
```



Technical B Skills: Simple and appropriate updating algorithms of record values.

Testing

Overview

In this section, I will carry out a series of tabulated and indexed tests to ensure that all the final requirements documented at the end of my Analysis are met and fully functional. In the event of a test's partial success or failure, I will detail the fixes to the codebase made to ensure that the test is passed. Additionally, I will include any general refinements/improvements made after testing that overall improve the experience of using the system. The testing table will be accompanied by videos split into multiple parts with screencast footage of my system. I will index tests with a timestamp using the minutes:seconds convention as well as which video it is in. This will show where in the video the test is carried out and which of the multiple videos it is in.

Some requirements were already tested in my Design section as prototypes, and so I will be referencing them with page numbers rather than timestamps. Furthermore, some tests do not require any input data (e.g. testing that all the obstacles are present within the visualisation), but rather require certain actions to be performed (pressing a button, scrubbing a slider, or simply just observing elements of the system and how they change).

Many of the tests are not explicitly derived from my requirements but are implied by them. For instance, I test the functionality of the genetic algorithm by individually testing its components separately from the main functioning of the system. Furthermore, some tests are carried out to ensure the robustness of the system, e.g., for handling any exceptions correctly.

It should also be noted that, for some tests, it is more conducive to use the Unity Editor over a compiled build of the game to test my requirements. This allows me to fully test some of the requirements that could only be simulated in the Unity Editor (e.g. testing the Quit functionality on pg. 70 of my Design). I will specify both in my Table and in the video whether a given test was carried out in a build or in the Editor.

Test Plan

Test #	Build or Editor?	Scene	Purpose	Test Data/Action	Expected Result	Test Type	Result	Video Part and Timestamp/ Page Number
1	Build	StartMenu	Check if user is first met with Main Menu	I will launch the .exe file of the build.	Upon launching the build, the user should be met with a menu containing 'Start', 'Options', and 'Quit' Buttons	Normal	Pass	Part 1, 0:50
2	Build	StartMenu	Check if the Music Volume slider in the Options Menu works	I will slide the Volume slider all the way down to 0.	The application should be muted.	Normal	Pass	Part 1, 0:50
3	Build	StartMenu	Check if user can quit out of the	I will press the 'Quit' button	The application should close,	Normal	Pass	Part 1, 2:42

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College. Centre Number: 74615. Component Code: 7517/C.

			application	on the Main Menu	and I should be met with my desktop.			
4	Build	StartMenu	Check that non-number mutation rates cannot be entered.	I will enter various erroneous inputs, like 'Taimur', '#@\$@', 'fifty two', 'True', whilst inputting a valid population size. I will then hit the 'Go' button.	The application should redirect me back to the Main Menu.	Erroneous	Pass	Part 1, 3:04
5	Build	StartMenu	Check that the correct boundaries for mutation rate are implemented.	I will enter integers and floats less than 0 or greater than 1, then enter 0, then enter 1. For each, I will enter a valid population size, then hit 'Go'.	The application should redirect me back to the Main Menu for all inputs except for the last two, in which case the main visualisation should start.	Boundary	Pass	Part 1, 4:42
6	Build	StartMenu	Check that non-integer population sizes cannot be entered.	I will enter various erroneous inputs, like 'Taimur',	The application should redirect me back to the Main Menu.	Erroneous	Pass	Part 1, 6:44

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College. Centre Number: 74615. Component Code: 7517/C.

				'#@\$@', 'fifty two', 'True', whilst inputting a valid mutation rate. I will then hit the 'Go' button.				
7	Build	StartMenu	Check that the correct boundaries for population size are implemented.	I will enter integers less than 2 or greater than 50, then enter 2, then enter 50. For each, I will enter a valid mutation rate, then hit 'Go'.	The application should redirect me back to the Main Menu for all inputs except for the last two, in which case the main visualisation should start.	Boundary	Pass	Part 1, 7:48
8	Editor	MainScene	Check that the correct number of agents are running, and generation number is initialised to 0	I will note what is on the screen by counting how many Agent GameObjects there are and looking at the Stats for this test.	The number of agents running should be equal to the value of population size entered in the previous menu. The 'Generation Num' field of the Stats should read 0.	Normal	Pass	Part 1, 10:50
9	Editor	MainScene	Check that all obstacles are	I will let the visualisation	There should be random	Normal	Pass	Part 1, 13:05

			present.	play out for an arbitrary length of time, until I can confirm the presence of spikes, boomboxes, and enemies.	number of spikes, boomboxes, and enemies throughout the course. As the visualisation runs for longer, the frequency of the obstacles should theoretically get more evenly distributed.			
10	Editor	MainScene	Check that Max Fitness statistic is working properly.	The visualisation will play out normally, and I will point out any updates made to the Max Fitness Stat.	When a given agent performs better than any other previous agent, the Max Fitness statistic should be updated.	Normal	Fail	Part 1, 14:23
11	Editor	MainScene	Check that Fitness Graph is working correctly.	The Debug console will output average fitness for a given generation, and I will observe whether the trend in values	The points plotted on the graph should correspond to the actual average fitness values.	Normal	Pass; could be refined	Part 1, 16:23

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College. Centre Number: 74615. Component Code: 7517/C.

				is reflected in the graph.				
12	Editor	MainScene	Check that the Generation Number Stat is working properly.	The Debug console will output the true Generation Number, and the displayed Gen Number will be observed.	As the generations progress, the Stat should be incremented.	Normal	Pass; could be refined	Part 1, 19:49
13	Editor	MainScene	Check that the Agents Remaining Stat is working properly.	I will pause the Scene, count how many agents are left, and compare this to the Stat.	The Stat should correctly denote how many agents are left running in the current generation.	Normal	Pass	Part 1, 21:57
14	Editor	MainScene	Check the Mutation Rate Stat is correct.	The Debug console will output the true mutation rate, which I will compare with the displayed rate.	The Stat should correctly denote the user-selected mutation rate.	Normal	Pass	Part 1, 23:56
15	Editor	MainScene	Check that the Reset Evolution Button works properly.	I will press the Reset Evolution button.	The entire scene should be reloaded with a new population, cleared graph	Normal	Pass	Part 1, 25:46

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College. Centre Number: 74615. Component Code: 7517/C.

					and stats and the generation counter reset.			
16	Editor	MainScene	Check that the Reset Evolution Button works on the first generation.	I will press the Reset Evolution button when the visualisation first begins.	The same result as Test 15.	Boundary	Pass	Part 1, 25:46
17	Editor	MainScene	Check that the End Evolution Button works properly.	I will press the End Evolution Button.	I should be redirected back to the Main Menu	Normal	Pass	Part 1, 28:00
18	Editor	MainScene	Check that after the End Evolution button has been pressed, system still functions as expected.	After pressing the End Evolution button, I will navigate to the Options Menu, then go back, then start the visualisation.	There should be no unexpected occurrences when navigating the system after ending an evolution	Normal	Fail	Part 1, 29:24
19	Editor	MainScene	Check that once all agents of a generation are dead, a new generation instantly starts.	I will let the visualisation carry out for 10 generations.	Upon the death of each generation, the new generation should begin instantly with a new population.	Normal	Pass	Part 1, 31:00
20	Editor	MainScene	Checking that the NNs have the	<i>See page reference</i>	<i>See page reference</i>	Normal	Pass	Pg. 83-85

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College. Centre Number: 74615. Component Code: 7517/C.

			correct structure.					
21	Editor	MainScene	Checking that theFeedForward method works properly.	<i>See page reference</i>	<i>See page reference</i>	Normal	Pass	Pg. 89-90
22	Editor	MainScene	Check that NN weight matrices are correct dimensions, and each weight is in the correct range.	<i>See page reference</i>	<i>See page reference</i>	Normal	Pass	Pg. 84-85
23	Editor	MainScene	Check that Sigmoid activation function is implemented properly.	<i>See page reference</i>	<i>See page reference</i>	Normal	Pass	Pg. 85-87
24	Editor	MainScene	Check that Tanh activation function is implemented properly	The Debug console will output values of the Tanh function for specific inputs. These inputs are -infinity, 0, +infinity	The Debug console should output a matrix [-1, 0, 1] in that order.	Normal	Pass	
25	Editor	MainScene	Test the SensorSystem's ability to detect obstacles, and	<i>See page reference</i>	<i>See page reference</i>	Normal	Pass	Pg.76-78

			the correct obstacles at that.					
26	Editor	MainScene	Ensure anAgent's NN is receiving inputs from its SensorSystem.	<i>See page reference</i>	<i>See page reference</i>	Normal	Pass	Pg. 94-95
27	Editor	MainScene	Test the normalized fitness values as an accurate and suitable measurement of an agent's performance	Once the curent generation is dead, the Debug console will output a sequence of normalized fitness values of the generation.	The values should all be between 0 and 1 and should suitably reflect the performance of each agent.	Normal	Pass (min-max normalization algorithm is as accurate as possible)	
28	Editor	MainScene	Check that all generations are of the same size (the population size)	Once a new generation starts, I will count the Agent GameObjects that spawned.	The number of new Agent GameObjects spawned should be equal to that of the previous generation, which in turn should be equal to the user-defined population size.	Normal	Pass	
29	Editor	MainScene	Check that the elitism property of the GA is	I will check whether the top two agents in	There should be two agents in the current	Normal	Fail	

			being carried out correctly.	the previous population are transferred directly to the current generation by comparing their neural network content.	population with weight matrices equal to those of the top two agents in the previous generation (i.e. they are the same).			
30	Editor	MainScene	Check that the two parents selected during the selection stage can be one, both, or neither of the two agents preserved due to elitism.	For each of a large number of parent selections, I will compare the weight matrices of the two selected parents and check whether either of them is also the elitism agents	Eventually, all three possible scenarios should occur; neither, one, or two of the parents also happen to be elitism agents.	Normal	Fail	
31	Editor	MainScene	Check that an agent can be selected twice during parent selection.	The visualisation will play out, and during a particular parent selection stage, the Debug console will output	After an arbitrary number of generations, there should be an instance where the required testing criteria is met	Normal	Pass	

				when the selected parents are both the same agent.	(an agent is selected twice as a parent).			
32	Editor	MainScene	Check that the alternating crossover function is working properly.	When two parents have been selected, The Debug Console will output Parent 1's weights, followed by Child 1's weights, followed by both the same pattern for Parent B and Child B.	One child's weight should have odd-indexed weight matrices from Parent 1, and hence even-indexed weight matrices from Parent 2. The other child should have the reverse.	Normal	Fail	
33	Build	MainScene	Check that the visualisation can handle a large number of reset requests in a short period of time.	I will press the Reset Evolution button repeatedly for 5 seconds.	The visualisation should continually be resetting back to its starting point (with new populations each time).	Normal	Pass	Part 1, 9:16
34	Editor	MainScene	Check that the GA mutates	<i>See page reference</i>	<i>See page reference</i>	Normal	Pass	Pg. 104-110

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College. Centre Number: 74615. Component Code: 7517/C.

			agents with probability equal to the mutation rate.					
35	Editor	StartMenu	Check that a GameManager object can't exist in the StartMenu scene.	I will create a GameManager object in the StartMenu scene, when normally it should only exist in MainScene.	The GameManager object should be destroyed immediately, thus preventing any error from occurring.	Normal	FAIL and could be refined to just make Manager check what's scene it's in and make it able to manage both instead of having MainMenu manage first scene). Destroy(object) instead of LoadScene.	
36	Editor	MainScene	Ensure that Generation Number can't be negative	I will set the starting generation number to be various negative numbers: -1, -5, and -100.	The generation number should still initialise to 0 on start of the visualisation every time.	Normal	Pass	
37	Build	MainScene	Check that the visualisation can run for a large number of generations.	I will leave the system running until it reaches 500 generations.	The system should run as normal, with no crashes or errors.	Normal	Pass	

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College. Centre Number: 74615. Component Code: 7517/C.

38	Editor	MainScene	Check that the array normalisation for fitnesses works properly.	The Debug console will output a list of fitnesses, followed by the list after passing it into the normalisation algorithm.	Higher fitnesses should map to values closer to 1, and vice versa.	Normal	Pass	
39	Build	MainScene	Test my <u>Success Criteria</u> (tests the relative success of the Investigation)	I will leave the system running for an indefinite number of generations	Eventually, there should be a generation where at least one agent could last longer than 28 seconds without dying.	Normal	Fail	

Testing Evidence

The evidence to the tests that were not already carried out in the Design section can be found split into parts at the following links:

PART 1 (Tests 1-19, 33): <https://youtu.be/8uX2zGo-xVQ>

PART 2 (Tests 24, 27-32, 35; Redo of Test 31 needed): <https://youtu.be/uzlsMOa-3WA>

PART 3 (Tests 36 and 38): <https://youtu.be/-q2D57vI6SM>

PART 4 (Timelapse for Tests 37 and 39): <https://youtu.be/9tmt5oH9rTA>

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College. Centre Number: 74615. Component Code: 7517/C.

PART 5 (Redoing Test 31): <https://youtu.be/UKANydyjsis>

Success Criteria Results

As you will see in Part 4 of the Testing Evidence, the algorithm was able to function for 500+ generations, so Test 37 was a pass. The final test, Test 39, involved finding an agent that survived for longer than 28 seconds. Across all 500+ generations, no agent was able to meet or beat this threshold. The closest agent observed just about reached 18 seconds. However, there were instances where this a clear increasing trend in the Fitness Graph, as well as evidence that generations are being influenced by previous ones. For example, a generation with lots of sliding agents will result in more and more sliding agents in the future generations. Generally, behavioural traits are inherited across generations. Whilst the Success Criteria failed, there is promising support for the fact that the fundamentals of what I tried to implement are present, albeit they are not utilised to their fullest potential.

After further consideration, it was unsurprising that the required behaviour was not met after only 500 generations. By examining similar projects (most of which I found from the channel Two Minute Papers*), I have seen that complex emergent behaviour similar to the behaviour required by agents in this project sometimes takes millions of generations to be achieved. Time and resource constraints certainly would not permit me to achieve this result. Regardless, somewhat promising results were observed when testing the system. Refinements and bug fixes are needed to make it the most effective it can be, however.

Refinements

In this section, I will go through the refinements required to pass the tests that were fails or partial successes.

Test 10:

This was a simple fix. First, we can define a maxFitness field in GAManager, and then update that anytime we encounter a fitness greater than any we have seen before.

```
public float maxFitness = 0f;
```

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College. Centre Number: 74615. Component Code: 7517/C.

```
if (res > maxFitness) {  
    maxFitness = res;  
}
```

Test 11:

There is potential for a tooltips feature to be implemented, where hovering over a point on the graph will display a small box displaying the actual (x, y) pair of generation number and average fitness. However, doing so may make the system a little convoluted to use, as there already is a considerable amount of screen real estate used up. Hence, I will consider this refinement out of scope.

Test 12:

This could simply be refined by initialising the generation number to 1 rather than 0, as users don't normally think of zero-indexing as their natural assumption.

```
GenerationNum = 1;  
private void Update()  
{  
    if (GenerationNum < 1) {  
        GenerationNum = 1;  
    }  
}
```

Test 18:

The problem here was that, once we are redirected back to the Main Menu after pressing the End Evolution Button, the AudioManager.cs script loses its reference to the themeMusic Audio object. To rectify, the OptionsMenu script needed to be tweaked to include the OnEnable() and OnDisable() MonoBehaviour Methods rather than Start(). OnEnable() runs everytime the GameObject is set to Active, and the opposite is true for OnDisable().

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College. Centre Number: 74615. Component Code: 7517/C.

```
public class OptionsMenu : MonoBehaviour
{
    private Slider musicSlider;
    private AudioManager am;
    private Sound themeMusic;
    private HoldValues hold;
    private void OnEnable()
    {
        musicSlider = GameObject.Find("MusicVolumeSlider").GetComponent<Slider>();
        am = GameObject.Find("AudioManager").GetComponent<AudioManager>();
        themeMusic = Array.Find(am.sounds, sound => sound.name == am.musicName);
        hold = GameObject.Find("Holder").GetComponent<HoldValues>();

        musicSlider.value = hold.MusicVolume;
    }

    private void OnDisable()
    {
        hold.MusicVolume = musicSlider.value;
    }

    // Executed on change of music slider
    public void UpdateMusicVolume()
    {
        // Set value of music volume to music slider value
        am.UpdateVolume(themeMusic, musicSlider.value);
    }
}
```

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College. Centre Number: 74615. Component Code: 7517/C.

We also need to make use of the Holder GameObject which preserves variable values between the two scenes. In this case, it was used to store the value of the music volume, so that when we revert back to the StartMenu, the music slider is at the same point where we left it.

```
public float MusicVolume
{ get; set; }

public static HoldValues instance;
void Awake()
{
    if (instance == null)
        instance = this;
    else
    {
        Destroy(gameObject);
        return;
    }
    GenerationNum = 1;
    MusicVolume = 0.5f;
    DontDestroyOnLoad(gameObject);
}
```

Test 27:

What I was unaware of during the testing stage was that the normalisation algorithm I used (the min-max algorithm) will **always** have the minimum value map to 0 and the maximum map to 1. Thus, my previous suggestion of improved accuracy was actually not an issue, so this test can be seen as a pass and not in need of any refinements.

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College. Centre Number: 74615. Component Code: 7517/C.

Test 29:

When first implementing this feature, I miscalculated my index math, and believed that I could not implement two-agent elitism with odd populations. By simply reconfiguring how I am accessing the agents, I am able to do this relatively easily regardless of whether the population size is odd or even. This is done through an offset, which ensures that the parent selection and crossover loop occur one extra time in the case of an odd population, thus preserving the same population size. Moreover, elitism now takes into account the fitness score itself rather than just adding the two agents that died most recently, as these agents are not guaranteed to have the highest fitnesses just because they lasted the longest. It is possible they excessively jumped or slid, and so had lower scores.

```
// Runs once all of this generation's agents are dead
public void EndGeneration()
{
    nextGen.Clear();

    float maxFitness = Fitnesses.Max();
    // NN of Agent with max fitness
    NN elitism1 = MatingPool[Fitnesses.IndexOf(maxFitness)];

    // Find NN of Agent with second largest fitness
    float temp = 0f;
    foreach (float f in Fitnesses) {
        if (f > temp && f < maxFitness) {
            temp = f;
        }
    }
    NN elitism2 = MatingPool[Fitnesses.IndexOf(temp)];
    nextGen.Add(elitism1);
    nextGen.Add(elitism2);
}
```

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College. Centre Number: 74615. Component Code: 7517/C.

The rest of the EndGeneration() method is the same as what is seen in my Technical Solution.

Test 30:

The conditional logic for this test did not consider all possible scenarios. The updated if statement is shown below.

```
if ((parentA == elitism1 && parentB == elitism2) || (parentA == elitism2 && parentB == elitism1)) {  
    Debug.Log("BOTH");  
} else if (parentA == elitism1 || parentB == elitism1 || parentA == elitism2 || parentB == elitism2) {  
    Debug.Log("ONE");  
} else {  
    Debug.Log("NEITHER");  
}
```

This, along with the fix shown in Test 29, resulted in the functionality working as expected, thus making Test 29 a pass.

Test 32:

The code that was written to carry out this test is below.

```
Debug.Log("PARENT A");  
for (int j = 0; j < parentA.weights.Count(); j++) {  
    if (j % 2 == 0) {  
        Debug.Log(parentA.weights[j]);  
    }  
}  
Debug.Log("CHILD A");  
for (int j = 0; j < childA.weights.Count(); j++) {  
    if (j % 2 == 0) {  
        Debug.Log(childA.weights[j]);  
    }  
}
```

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College. Centre Number: 74615. Component Code: 7517/C.

```
Debug.Log("PARENT B");
for (int j = 0; j < parentB.weights.Count(); j++) {
    if (j % 2 != 0) {
        Debug.Log(parentB.weights[j]);
    }
}
Debug.Log("CHILD B");
for (int j = 0; j < childB.weights.Count(); j++) {
    if (j % 2 != 0) {
        Debug.Log(childB.weights[j]);
    }
}
```

I misinterpreted what the code was actually outputting. For Parent A and Child A, the code is outputting every **even-indexed** weight matrix (so matrix 0 and 2). For Parent B and Child B, the code is outputting every **odd-indexed** matrix (matrix 1 and 3). So, what we actually should expect to see is that the 2 matrices listed below Parent A should be **the same as** those of Child A. The same applies for Parent B and Child B. This is actually what we saw in the test video, but I misinterpreted the values. Here is another example of the test working in action.

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College. Centre Number: 74615. Component Code: 7517/C.

```
[11:34:28] PARENT A
UnityEngine.Debug:Log(Object)
[11:34:28] DenseMatrix 10x8-Single
0.0928062 0.46974 0.886934 0.530294 -0.116605 .. 0.280569 0.912492
[11:34:28] DenseMatrix 8x8-Single
0.621286 0.825632 -0.627032 0.719091 -0.748269 .. -0.184554 0.60023
[11:34:28] CHILD A
UnityEngine.Debug:Log(Object)
[11:34:28] DenseMatrix 10x8-Single
0.0928062 0.46974 0.886934 0.530294 -0.116605 .. 0.280569 0.912492
[11:34:28] DenseMatrix 8x8-Single
0.621286 0.825632 -0.627032 0.719091 -0.748269 .. -0.184554 0.60023
[11:34:28] PARENT B
UnityEngine.Debug:Log(Object)
[11:34:28] DenseMatrix 8x8-Single
0.590953 -0.0429558 0.027316 0.567025 0.548389 .. -0.765238 -0.0630785
[11:34:28] DenseMatrix 8x5-Single
0.334437 -0.914882 0.366296 0.695025 -0.590254
[11:34:28] CHILD B
UnityEngine.Debug:Log(Object)
[11:34:28] DenseMatrix 8x8-Single
0.755574 0.209932 -0.0225655 -0.905059 -0.289674 .. -0.109066 0.698626
[11:34:28] DenseMatrix 8x5-Single
0.000350595 0.360807 0.485568 -0.0768148 0.641467
```

Test 35:

Whilst a cleaner, more elegant solution can definitely be achieved, doing so would require considerable upheaval of the internal functionality of my system. As such, the following refinement works effectively, albeit a little clumsy. In my initial code, I did not actually destroy the GameManager object after checking if we are on the StartMenu scene. So now I have simply fixed that in the code below. If we are not on the StartMenu scene, we can get the required components without the risk of an error, because we are guaranteed to have those objects present in the MainScene.

```
if (SceneManager.GetActiveScene().buildIndex == 0) {
    Destroy(gameObject);
} else {
    genetic = GameObject.Find("GAManager").GetComponent<GAManager>( );
    hold = GameObject.Find("Holder").GetComponent<HoldValues>( );
}
```

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College. Centre Number: 74615. Component Code: 7517/C.

```
graph = GameObject.Find("WindowGraph").GetComponent<WindowGraph>();  
}
```

Test 39:

See the [Success Criteria Results](#) for an explanation of this test.

Additional Refinement: Improved Fitness Function

In retrospect, the Fitness Function could be altered to better mathematically represent the performance of the agent. I have settled on the updated formula:

$$F(a) = d - \phi (j+s)$$

Where a is an agent, d is the agent's z position, j is the jump count, s is the slide count, and ϕ is some multiplier (usually very small). For the refinement, I have set it to be 0.01.

```
const float fitnessMult = 0.01f;  
  
public float Fitness(Transform agent, float jumpCount, float slideCount)  
{  
    float res = agent.position.z;  
    if (jumpCount > 0 && slideCount > 0) {  
        res -= (fitnessMult * (jumpCount + slideCount));  
    }  
    return res;  
}
```

Evaluation

Individual Objectives

This table restates my overarching Final Requirements. Note that some have been abbreviated for convenience and readability.

No.	Objective	Evaluation
1	Upon start-up of the system, the user must see a start menu, where they must click a Start button in order to continue.	<p>Completely Achieved: My menu has a straightforward design, with a responsive UX: the buttons darken when pressed, thus giving a clear response to the end user. Mapping out simple wireframes and mock-ups in my Analysis helped to give me a very clear picture of what should be on the screen when it came time to translate the design into Unity.</p> <p>Whilst the objective is achieved, the menu itself could be improved in both aesthetics and functionality. In one of my first discussions with my Supervisor, I did specify that aesthetic appeal was not a primary focus of the project (see Meeting 1). However, now I see that I could have still put some degree of thought into it, as it would make for an overall better experience for an end user. It is currently only incorporating bland shades of grey and black, which clash with the colourful neon hues in the main visualisation. Additionally, auditory or enhanced visual feedback upon interaction with the menu could make for a better user experience, as well as once again more nicely coinciding with the aesthetic of the main system. Mr. Wood did suggest this improvement in his feedback by suggesting “Perhaps there is room for a more exciting design?”</p>
2	After starting the game, the user must be prompted to enter two parameters that will be passed as parameters into the genetic algorithm	<p>Completely Achieved: The parameter input screen functions as expected – it is shown directly after the user presses ‘Start’ on the Main Menu and contains two clearly labelled text fields (‘Mutation Rate’ and ‘Population Size’). The idea</p>

		<p>behind this feature was to allow the user to have an element of control over the functioning of the AI, as these two parameters are pivotal to its success in its environment.</p> <p>That being said, this objective, whilst fully implemented, achieves the intended purpose as well as it could. This is because there is no initial explanation of what these two parameters control, nor is there an explanation of why they are important. Mr Wood expressed this issue in his feedback as well, stating that this screen confused him. Having some kind of disclaimer may help an unfamiliar end user make their choice on which parameter set to choose for a particular session.</p>
3	The parameters entered in Requirement 2 must be validated	<p>Partially achieved: The validation algorithm is correctly implemented; in that it only accepts mutation rates and population sizes within the allowed range. If either of the fields is populated with an invalid input, then the user is redirected back to the Start Menu.</p> <p>I feel as though this method of validation is somewhat incomplete from the user's side, as there is no form of feedback on what was wrong with their input. This could potentially lead to confusion which may cause the user to just resort to trial and error of different inputs until a valid one is entered. This defeats the whole purpose of Requirement 2, so it is something that needs to be rectified should the system be publicly released.</p>
4	After validation of parameters in Requirement 3, the user must confirm their choices, upon which the visualisation starts. All the features stated in my Final Requirements should be present in this visualisation.	<p>Partially Achieved: As stated in my Technical Solution, Requirement 4d could not be met due to technical limitations; speeding up the game for a prolonged period of time puts an enormous amount of strain on the CPU and GPU, especially with the amount of matrix calculations occurring behind the scenes in both my NN scripts and Unity's 3D Rendering system. I'm sure that with significant optimisation to my code, I would be able to implement a simplified version of the feature. One idea suggested by my schoolmate was the concept of 'internally' evolving the AI. This will be elaborated upon under the <u>Further</u></p>

		<p><u>Improvements</u> heading.</p> <p>All other visual features I set out to include are indeed present in the final build. I did end up having to alter some of the code from the base ER prototype. These were primarily optimisations and general refinements that allowed the AI system I built on top of this existing codebase to integrate more smoothly. The most alterations were made to the scripts governing player movement, collision, camera control, and overall management of the game. This process did lead to features breaking along the way, resulting in continuous refinements being made to the new system being added as well the existing scripts. It may have been more streamlined to start from scratch, as it would ensure every script was written with the final intention of being integrated with this final project, AI included. Regardless, I think the structure of this project is overall sound, albeit with clunky design at a few points.</p>
5	A custom Neural Network class must be defined.	<p>Completely Achieved: I believe that the <u>NN.cs</u> class I created works well for its intended purpose. Although my use of the OOP paradigm throughout this project was simple and limited, this class incorporates many of the desirable features that make OOP desirable. For instance, the crucial Feed Forward function of the NN is encapsulated within the class, and unchanging values like the hyperparameters of the NN can be easily defined at the top of the script using static fields. Use of the MathNet.Numerics library also makes the NN's methods as efficient as possible given the scope of the project.</p>
6	An AI agent must be represented as a Neural Network object attached to an existing Player object which has a PlayerMovement script that was present in the base game as well as a 3D model.	<p>Partially Achieved: Abiding by this concept of an agent throughout the project's development vastly simplified the process; each of the constituent elements of an Agent was developed separately, then put together for testing. I find that a biological analogy works very well when imagining how this model is structured. The NN object is like the brain, which takes input from the eyes (the <u>SensorSystem.cs</u> script). The NN then causes the body to react to its environment (the 3D model which is controlled by the <u>PlayerMove.cs</u> script, among others). This</p>

		<p>intuitive model allowed me to program features more effectively for each 'body part', as I was constantly aware of their larger purpose.</p> <p>My initial idea of a CalculateFitness() method that is defined within a script attached to every agent ended up not being the most effective way of handling Fitness. I instead opted to include this method in the GAManager.cs script, as it made more intuitive sense to have all GA-related method contained in that one central script. This did not greatly affect the larger structure of my agent model. It may have helped to consider this early on in the Design phase, as it may have informed better decisions in the final build that could have led to less bugs and troubleshooting due to the structure of the project not making sense at points.</p>
7	Once all agents in a generation have been eliminated, the processes of the genetic algorithm are carried out to generate the new population.	<p>Completely Achieved: I am content with the actual way in which the GA was implemented; a master script (GAManager.cs) to handle all GA-related functionality made the most sense for this project. It also greatly simplified the OOP model, as almost everything in that script is public. Admittedly, this would be bad practice for a more large-scale project, but for a visualisation like this with relatively few scripts, it was the most straightforward option for a master script like that. Regardless, the script would definitely need to be heavily refactored should this system be extended upon in the future for a larger project.</p> <p>As will be discussed further, the actual effectiveness of the GA was suboptimal. Whilst there is clear evidence of learning occurring, it is not nearly fast enough to achieve the Success Criteria in a feasible amount of time. This could be due to a multitude of reasons. It's likely that the setup of the GA was not optimal for this specific purpose. For instance, I used an alternating crossover function, when perhaps a different variant of the crossover algorithm would've worked better. Even more, perhaps a crossover function was not needed at all – many GA applications operate well without this step. This, along with other factors, may have helped with improving the algorithm's subpar</p>

		performance.
8	The processes listed in Requirement 7 must be repeated indefinitely until the user chooses to end or reset the evolution.	Completely Achieved: I have fully implemented a 'game loop' of sorts, where the system is able to reset certain parameters and properties in preparation to start a new generation, or to reset or end the evolution when the user has pressed the appropriate button. Various tests from across my Testing section can attest to this. The actual design for this feature is something that can definitely be improved, as although these processes are largely contained to two scripts (GameManager.cs and GAManager.cs), I find the ways in which they interact with each other as well as the other elements that need to be reset to be somewhat convoluted at times. An example of this would be the use of default parameters when resetting. See Further Improvements for a more detailed explanation on how to combat this as well as potential trade-offs that may arise in doing so.

Overall Effectiveness

By conducting a thorough and expansive analysis at the start of my investigation, including initial consultations with my supervisor, supplemented with meetings with an expert in the field and a potential user, I was able to form requirements that covered most areas expected of a system like this. Specifically, the Algorithmic Requirements were extremely useful in solidifying my understanding the more complex mechanisms my project. I could always refer back to these objectives to refresh my memory when implementing a new subcomponent of the project's AI system. As a result, the NN and GA are both implemented in ways consistent with existing solutions and modern understanding of their effectiveness. This by no means implies that they are the most effective implementations for this specific context; I am simply stating that the required algorithms and structures are implemented conventionally and effectively. For instance, the use of matrix multiplication in the NN class is a useful optimisation and overall a cleaner practice in machine learning.

Despite this, my project is not without its shortcomings. The most glaring issue with this investigation is the fact that the proposed [Success Criteria](#) was not met. As such, I have no evidence to conclude that my AI system can be **at or above the level of a human player**. It is certainly possible that, with sufficient training, there exist agents that are truly as good as humans, but various limitations impeded the discovery of such agents. The two main problems that prevented this were the time needed to evolve a population that

contained a successful agent, as well as the device that my system is being run on. The laptop that I have run the system on, as well as written all the code on, is a mid-range laptop with 8GB of RAM. In preliminary testing of the basic functioning of the system, there were select instances where Unity would crash due to the GPU being overloaded. This was mainly when testing Requirement 4d (which I ultimately did not implement), but there were also other instances. Granted, these were all in the Unity Editor rather than a compiled build (in which this issue was not present) but is still unexpected and grounds for a potential optimisation.

Many potential requirements could not be implemented due to a multitude of reasons. An example of this is a visual representation of the NN belonging to the current best performing agent the top of the screen. In the [Consideration](#) of this objective, I decided not to include it due to resource constraints and to avoid overly convoluting the UI.

Additional issues and potential improvements have been mentioned in the table above. These include rethinking the GA and refining its constituents for this specific project to maximise effectiveness. Moreover, general UI, aesthetic, and quality-of-life improvements would make the project more polished.

Analysis of Feedback from Supervisor

In this table, my objectives are again listed. Accompanying each is feedback from my Supervisor, Mr. Mark Wood, directly quoted from his official feedback (see [Appendix](#)).

No.	Objective	Supervisor Feedback and Analysis
1	Upon start-up of the system, the user must see a start menu, where they must click a Start button in order to continue.	<p><i>“There is not much to say about the first requirement. It is implemented and it works in all instances and scenarios when I was using the system. Perhaps there is room for a slightly more exciting design? Additionally, if improvements were made to other aspects of the project, then there may be potential to include a degree of control for these features in the main menu where applicable.”</i></p> <p>Analysis: The first suggestion Mr. Wood made here is to do with the menu’s aesthetics, which I have already stated would be an actionable and effective update to make. I can gather from the way he is describing it that this is not a huge priority but would indeed contribute to the system’s appeal to the general user.</p>

		<p>The second part of Mr. Woods' feedback with regards to the menu gave me ideas on some of the potential additions to make given improvements to the other objectives. Some of these don't involve a 'degree of control' however. For example, it may make for a more coherent experience if the purpose of the mutation rate and population size parameters were made clearer on the parameter input screen. This could simply be implemented as static text, and so does not involve any additional input. Improvements that do involve user input include options for camera control as the agents are running, a checkbox to enable the use of the crossover function, or instead a select field for choosing the specific type of crossover function to use. Additions like these would not improve the system in other areas but would make for a less bare-bones menu, thus giving it a larger sense of purpose in the user's eyes.</p>
2	<p>After starting the game, the user must be prompted to enter two parameters that will be passed as parameters into the genetic algorithm.</p>	<p><i>"Using the parameter input screen was confusing. The main reason for this was the lack of any sort of guidance on the kinds of input either field could accept. I could infer they both had to be some kind of numerical input, but the specific ranges and natures were not conveyed. When a supposed invalid input was entered, I was sent back to the Start Menu with no warning or indication of what was wrong with my inputs. It was only through explanation by yourself that I was able to understand a.) the purpose of the parameters, and b.) what their acceptable set of inputs were. I would highly suggest implementing additional UI features that eliminate any of this confusion."</i></p> <p>Analysis: In the preface to my Model User Meeting, I did state that I intend for my system to be accessible to users with varying degrees of knowledge about GAs. As a result, the confusion Mr Wood experienced (who is not familiar with GAs) is a clear indicator of the need for this objective to be added to. As mentioned in the previous row, including static text explaining the purpose of population size and mutation rate as</p>

		<p>well as their valid ranges of inputs would be the first step, and is relatively simple to do. From there, incorporating specific error messages related to the user's input would be the logical next step. The number of unique messages will be discussed under the next heading.</p>
3	The parameters entered in Requirement 2 must be validated.	<p><i>"After gaining an understanding of the nature of valid inputs, I was able to verify by exhaustion that all invalid inputs (if not, then the vast majority) are rejected by the menu."</i></p> <p>Analysis: The quote from Mr Wood's feedback for the previous objective can be applied here alongside the current one. The validation algorithm itself cannot be viewed by the user, but by verifying that only valid inputs are accepted, Mr Wood was able to confirm the success of this objective. The improvement suggested in Requirement 2 that proposes specific error messages indirectly applies to this one as well.</p>
4	After validation of parameters in Requirement 3, the user must confirm their choices, upon which the visualisation starts. All the features stated in my Final Requirements should be present in this visualisation.	<p><i>"You and I did discuss what was already present in the base model of the game last year during the Analysis phase, and I was able to see every aspect I saw there in this finished build, now with an AI system fitted on top to play the game. The aesthetics and colour palette of the game itself are appealing, and everything on the screen is easily distinguishable. Each agent can clearly be seen running, jumping, and sliding around and it does appear that they all have behaviours of their own."</i></p> <p><i>Performance issues seemed to be the biggest hindrance to the functioning of the system; the frame rate was wildly inconsistent at times, and it appeared as though there were minor glitches where some runners appeared to pass through obstacles. The frame rate issues were most prevalent when a new generation of runners was spawning in. As for the glitches, they were few and far in between –in about 10 runs of the system, each of which I evolved to 100 generations, I only spotted 2 of these glitches. Regardless, it is definitely worth taking a look at if my understanding of Genetic</i></p>

		<p><i>Algorithms is correct.”</i></p> <p>Analysis: Confirming the presence of all the existing obstacles and objects in the base model was practically impossible to do had I not shown Mr Wood the prototype back in December 2020 (it is January 2022 as of writing this section). Indeed, the spikes, boomboxes and enemies that I initially showed him were still functioning as before in this finished build. Additionally, the terrain of the course tiles looked the same. The only substantial difference is the adjustment in camera placement to be farther above the course to accommodate for there being multiple agents running at a time as opposed to one.</p> <p>Mr Wood is absolutely correct in highlighting the issues caused by the framerate. When a new generation spawns, especially when a high population size has been selected, it can at times feel like the user is looking at a set of still images before the framerate stabilises. I can assume this is due to excessive strain on the GPU for rendering up to 50 new 3D models all at the same time that causes this. It may mitigate the issue to incorporate some kind of batching procedure, whereby the population size is divided into chunks, and each of these chunks is rendered sequentially. If implemented correctly, then this could drastically improve performance whilst preserving the original spawn time. A batching procedure may also help with the few Physics glitches Mr Wood observed. Thinking from the first principles of my system, this occurs because a collision was not registered between the Agent's and the obstacle's Collider components, which can only be caused by too many calculations at once. By breaking these calculations into groups of calculations and then executing them very quickly one-by-one, I believe that the issue can be improved, if not eliminated entirely.</p>
5	A custom Neural Network class must be defined.	<p><i>“Your overview of how you implemented a neural network class following OOP principles was very insightful. The class is robustly programmed and makes good use of encapsulation. The use of a 3rd party library for some of the methods does help with making the code cleaner and easier to</i></p>

		<p><i>understand. I would imagine that implementing everything from scratch would be unnecessary and waste time in the development process. Judging by what can be observed in the visualisation, it appears as though the neural networks of each runner is functioning effectively, as runners tend to change their current behaviour if at least one of their five sensors detects an obstacle.”</i></p> <p>Analysis: Again, this requirement would have been impossible to verify if I hadn’t shown Mr Wood the NN model itself, and then explain how it fits into the overall system. When he mentions the sensors, he is referring to the green Debug lines that represent the sensors in SensorSystem.cs. As a reminder, these lines are only visible in the Unity Editor, not the final build. Mr Wood did not mention any specific improvements for this requirement, but it has got me thinking about how I may flesh out the visual representation of the Sensor System. It’s possible that there is potential to have the lines visible in the final build, although through different means (no Debug outputs can be viewed in a compiled build). Implementing this may give a user who is using the final build more insight into how each agent is ‘deciding’ its next move.</p>
6	An AI agent must be represented as a Neural Network object attached to an existing Player object which has a PlayerMovement script that was present in the base game as well as a 3D model.	<p><i>“I appreciated you showing me how all the pieces of an agent fit together. The decision to model split up the neural network, movement and sensor handling into separate scripts at first seemed unnecessary, as at their core they were all manipulating a given runner, so why not just have them all as one ‘Runner.cs’ script? As I understood more of Unity’s conventions, as well as how the overall system model functioned, I grasped that this in fact was a fairly effective design choice. I did find the way in which the ‘AgentNN’ script was implemented to be rather clunky, as it is almost an intermediary script between the Neural Network class and the runner’s movement script. This seemed unnecessary to me, as the network influencing a runner’s movement could have easily been placed in the movement script and made for an overall cleaner design.”</i></p> <p>Analysis: I did find the process of making</p>

		<p><u>AgentNN.cs</u> script to not be idiosyncratic to C#. This should have been an indication to take a different approach. Having public references to both a given agent's NN as well as their movement script partly defeats the purpose of the OOP paradigm I followed. Granted, the model I followed was simplified, and as a result there are other select instances in which unorthodox decisions are made. Reworking the codebase to strictly follow a Unity-centric OOP paradigm may prove for more elegant implementations as well as the improved practicality that this strict OOP allows.</p>
7	<p>Once all agents in a generation have been eliminated, the processes of the genetic algorithm are carried out to generate the new population.</p>	<p><i>"If left untouched, the program continually resets itself once all runners in a generation are dead. There is clear indication that runners are somewhat inspired by previous generations, given how they react to specific situations. For instance, I noticed during one session that a lot of runners across generations continually gravitated towards sliding when they saw spikes directly in front of them. This is obviously not the correct decision, but if my understanding is correct, it does provide evidence for the genetic algorithm functioning as expected in this early stage of its runtime. When conversing with you, you were able to give me an overview of how this genetic algorithm works, including an explanation of how each method contributed to the overall solution to the optimisation problem. Judging by how progress as generations pass can be observed, I am confident in saying that this objective has been met.</i></p> <p><i>I am aware that your criteria for success was not met, and I imagine a large part of the blame could be attributed to the algorithm itself, as this is what drives the evolution. Whilst I can't provide any specific suggestions on what to look at, it is likely worth examining and experimenting with each component to find out what the issue was."</i></p> <p>Analysis: The comments Mr Wood made here, specifically on my Success Criteria more or less fall in line with <u>my own comments</u>. The fact that he was able to notice some form of behavioural inheritance across generations is very encouraging, as I presumed I would have some</p>

		<p>positive bias towards the system's effectiveness as its creator. The GA is indeed the most likely culprit for the failure of the criteria, and so tweaking it would be my first order of business if I were to attempt to meet the criteria. As mentioned previously, I would first examine my implementation of crossover, and consult scholarly research on its use cases, and come to a conclusion on the overall changes I should make to the algorithm.</p>
8	<p>The processes listed in Requirement 7 must be repeated indefinitely until the user chooses to end or reset the evolution.</p>	<p><i>"If left untouched, the program continually resets itself once all runners in a generation are dead. There is clear indication that runners are somewhat inspired by previous generations, given how they react to specific situations. For instance, I noticed during one session that a lot of runners across generations continually gravitated towards sliding when they saw spikes directly in front of them. This is obviously not the correct decision, but if my understanding is correct, it does provide evidence for the genetic algorithm functioning as expected in this early stage of its runtime. When conversing with you, you were able to give me an overview of how this genetic algorithm works, including an explanation of how each method contributed to the overall solution to the optimisation problem. Judging by how progress as generations pass can be observed, I am confident in saying that this objective has been met.</i></p> <p><i>By pressing the Reset button, it restarts from scratch. Aside from the counter resetting and agents respawning, there is no other clear visual indication of the reset. Some kind of UI element would make resetting a little clearer. As for ending the evolution, I think a confirm screen saying something 'Are you sure you want to end the evolution?' would prevent users from accidentally ending when they may have just wanted to reset?"</i></p> <p>Analysis: This is the first half of the same feedback used for Requirement 7. The continuous looping of the GA can be thought of as a GA process, so in reality this requirement is loosely related the previous one. I do agree that resetting does a feel a little lacklustre, and improved UI</p>

		<p>would allow the new population to feel fresh and exciting. In fact, improved UI and UX elements across the board would greatly enhance my system's appeal to users.</p> <p>Adding a confirmation screen before ending the evolution would likely be very easy to do, as it is basically just a UI box with two buttons, both of which can use Unity's EventSystem to run a simple script that calls the appropriate method. This does act as a method of fool proofing my system; as Mr. Wood said, it would prevent accidental terminations, and so perhaps save users some inconvenience of having to go through the Main Menu again when they just wanted to reset the evolution instantly.</p>
--	--	---

Further Improvements

Many of the potential improvements have been outlined in the previous sections, but I will reiterate them here and suggest a rough overview of the process by which they can be implemented.

When the user chooses to end an evolution, a simple confirmation screen verifying whether the user wants to leave to the main menu would reduce the likelihood of accidental exits due to a misclick (for instance, when the user wanted to press the Reset button). Menus such as those seen in video game menus (see below) would act as the main inspiration for the design and functionality of my own.



Figure 56: Sample Quit Screen from 'Clash Royale'

A big problem with the Parameter Input phase of the system in its current state is the lack of information given to the user about their inputs. This includes information about the inputs themselves (range of valid inputs, data types, brief explanation of what their

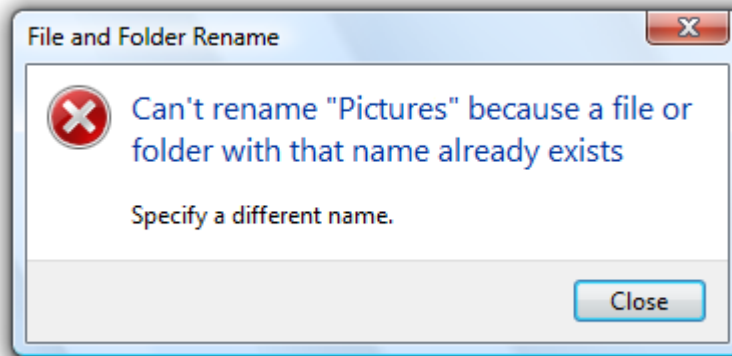


Figure 57: Sample Error Screen from Windows 8 OS

purpose is) as well as feedback on why an input was invalid. These messages could be delivered via a pop-up menu with an 'OK' button for when the user is finished reading the message. A great example of such error pop-ups can be viewed in the Windows operating system (see below). Alongside helpful error messages, having an explanation at the outset would also combat this issue. The parameter input screen could be re-designed with space for text boxes that provide a summary of what the purpose of the population size is, what inputs it can take etc., and the same thing for mutation rate. Another approach would be a button/other clickable that would cause a pop-up with the same information to be displayed. This would save on screen real estate on the main parameter input screen.

After further research, I discovered that the batching concept described in the previous sections is actually already done by Unity on similar meshes to save on rendering time. After learning this, I realised that there is not much else that can be done specifically with this technique to improve performance. Another area to attack could be general programming practices. Whilst my [Technical Solution](#) shows my use of effective programming practices, there are pockets in my codebase that include design decisions that negatively impact performance. These effects can stack up if they are too abundant. Thus, my strategy should involve combing through the densest scripts first and reworking any bad practices where possible. These include nested for loops, a linear search where a binary search may be quicker, faster sorting algorithms, less reliability on built-in methods that have a high time complexity.

General visual and quality-of-life improvements such as sharper text, better lighting settings and more professional post-processing would enhance the user experience significantly. Functionality for all of these aspects is well-implemented in Unity via

TextMeshPro, volumetric lighting, and Unity's own post processing system. Thus, tweaking them is not hard. Finding the perfect settings for all of them requires trial and error, as well as more experience than I currently have, as there are many different visual parameters that to be controlled, so it requires time to understand what they all do. Regardless, with time, I believe that it can be learnt. Additional visual improvements could be the addition of particles to the game using Unity's ParticleSystem. These, like the other visual aspects, can be tweaked heavily, allowing for the production for a wide variety of different particle effects. In my case, I'd likely need particles to do with high-speed movement to reflect the runners.

Finally, I could rework the codebase to essentially eliminate the need for [AgentNN.cs](#) by basically migrating all of its functionality to what is currently the [PlayerMove.cs](#) script. [NN.cs](#) then only needs to communicate with PlayerMove.cs. Not only would this make the code cleaner and the relations between scripts slightly more intuitive, but it would result in a performance boost in the Editor, as less scripts need to be compiled at runtime.

Appendix

Text from Mr Wood's feedback letter.

Dear Taimur,

First of all, I must say the way in which you have gone about completing this project is exemplary in terms of an investigation; it has been exciting to see you at each check point to find out what you have explored, experimented with and produced. It really was a true investigation. Please take this feedback as an honest opinion and reflection of the final outcome and please feel free to come and speak to me about, or make reference to, any of the points below.

In terms of the investigation purpose, this project seems to be a promising, yet still partial solution. Whilst all main components are implemented well and do in fact show that they are working towards the required goals, I understand that this goal could not be reached. That being said, this definitely is a fascinating and well put together system, with lots of potential for further development.

There is not much to say about the first requirement. It is implemented and it works in all instances and scenarios when I was using the system. Perhaps there is room for a slightly more exciting design? Additionally, if improvements were made to other aspects of the project, then there may be potential to include a degree of control for these features in the main menu where applicable.

Using the parameter input screen was confusing. The main reason for this was the lack of any sort of guidance on the kinds of input either field could accept. I could infer they both had to be some kind of numerical input, but the specific ranges and natures were not

conveyed. When a supposed invalid input was entered, I was sent back to the Start Menu with no warning or indication of what was wrong with my inputs. It was only through explanation by yourself that I was able to understand a.) the purpose of the parameters, and b.) what their acceptable set of inputs were. I would highly suggest implementing additional UI features that eliminate any of this confusion. After gaining an understanding of the nature of valid inputs, I was able to verify by exhaustion that all invalid inputs (if not, then the vast majority) are rejected by the menu.

You and I did discuss what was already present in the base model of the game last year during the Analysis phase, and I was able to see every aspect I saw there in this finished build, now with an AI system fitted on top to play the game. The aesthetics and colour palette of the game itself are appealing, and everything on the screen is easily distinguishable. Each agent can clearly be seen running, jumping, and sliding around and it does appear that they all have behaviours of their own.

Performance issues seemed to be the biggest hindrance to the functioning of the system; the frame rate was wildly inconsistent at times, and it appeared as though there were minor glitches where some runners appeared to pass through obstacles. The frame rate issues were most prevalent when a new generation of runners was spawning in. As for the glitches, they were few and far in between – in about 10 runs of the system, each of which I evolved to 100 generations, I only spotted 2 of these glitches. Regardless, it is definitely worth taking a look at if my understanding of Genetic Algorithms is correct.

Your overview of how you implemented a neural network class following OOP principles was very insightful*. The class is robustly programmed and makes good use of encapsulation. The use of a 3rd party library for some of the methods does help with making the code cleaner and easier to understand. I would imagine that implementing everything from scratch would be unnecessary and waste time in the development process. Judging by what can be observed in the visualisation, it appears as though the neural networks of each runner is functioning effectively, as runners tend to change their current behaviour if at least one of their five sensors detects an obstacle.

I appreciated you showing me how all the pieces of an agent fit together*. The decision to model split up the neural network, movement and sensor handling into separate scripts at first seemed unnecessary, as at their core they were all manipulating a given runner, so why not just have them all as one 'Runner.cs' script? As I understood more of Unity's conventions, as well as how the overall system model functioned, I grasped that this in fact was a fairly effective design choice. I did find the way in which the 'AgentNN' script was implemented to be rather clunky, as it is almost an intermediary script between the Neural Network class and the runner's movement script. This seemed unnecessary to me, as the network influencing a runner's movement could have easily been placed in the movement script and made for an overall cleaner design.

If left untouched, the program continually resets itself once all runners in a generation are dead. There is clear indication that runners are somewhat inspired by previous generations, given how they react to specific situations. For instance, I noticed during one session that a

lot of runners across generations continually gravitated towards sliding when they saw spikes directly in front of them. This is obviously not the correct decision, but if my understanding is correct, it does provide evidence for the genetic algorithm functioning as expected in this early stage of its runtime. When conversing with you, you were able to give me an overview of how this genetic algorithm works, including an explanation of how each method contributed to the overall solution to the optimisation problem. Judging by how progress as generations pass can be observed, I am confident in saying that this objective has been met.

I am aware that your criteria for success was not met, and I imagine a large part of the blame could be attributed to the algorithm itself, as this is what drives the evolution. Whilst I can't provide any specific suggestions on what to look at, it is likely worth examining and experimenting with each component to find out what the issue was.

By pressing the Reset button, it restarts from scratch. Aside from the counter resetting and agents respawning, there is no other clear visual indication of the reset. Some kind of UI element would make resetting a little clearer. As for ending the evolution, I think a confirm screen saying something 'Are you sure you want to end the evolution?' would prevent users from accidentally ending when they may have just wanted to reset?

Overall, as mentioned I think what you have been able to achieve in this project is outstanding. It really was and is a true investigation. Thank you and well done.

Mr. Wood

** : At various points during the project's Implementation phase, I showed Mr. Wood how some of the higher-level complexities worked by demoing the complexity or showing him the relevant code.*

Sources

CS50 Podcast Episode 6: [youtube.com/watch?v=mEdIQbOL8dY](https://www.youtube.com/watch?v=mEdIQbOL8dY)

Existing Solution 1: <https://github.com/LozzLol/endless-runner>

Research Paper on calculating game difficulty:

https://www.researchgate.net/publication/251710656_Measuring_the_level_of_difficulty_in_single_player_video_games

Flappy Bird: <https://flappybird.io/>

Existing Solution 2: <https://github.com/aayusharora/GeneticAlgorithms>

Existing Solution 2 Medium articles:

<https://heartbeat.fritz.ai/automating-chrome-dinosaur-game-part-1-290578f13907>

Candidate Name: Taimur Shaikh. Candidate Number: . Centre Name: Dubai College.
Centre Number: 74615. Component Code: 7517/C.

<https://heartbeat.fritz.ai/using-genetic-algorithms-to-automate-the-chrome-dinosaur-game-part-2-1c0007334297>

Existing Solution 3: <http://www.liampatell.org/endless-runner-neural-network/>

Additional Existing Solution (MarI/O):

<https://www.youtube.com/watch?v=qv6UVOQoF44>

MarI/O Source Code: <https://pastebin.com/ZZmSNaHX>

NEAT Research Paper: <http://nn.cs.utexas.edu/downloads/papers/stanley.eco2.pdf>

Video on NEAT: <https://youtube.com/watch?v=b3D8jPmcw-g>

Jeff Heaton Research Blog: <https://www.heatonresearch.com/2017/06/01/hidden-layers.html>

Two-point crossover: https://www.researchgate.net/figure/Two-point-crossover_fig2_220485962

Agent 3D model:

<https://assetstore.unity.com/packages/3d/characters/humanoids/lowpoly-ninja-157942>

Math.NET Numerics library: <https://numerics.mathdotnet.com/api/>

Chart Plotting GitHub Repo: <https://github.com/codemaker2015/Line-Graph>

Dynamic Line Chart Asset: <https://assetstore.unity.com/packages/tools/gui/dynamic-line-chart-108651>

SFXR: <https://sfxr.me/>

Daniel Shiffman's Parent Selection Algorithm: <https://www.youtube.com/watch?v=-jv3CgDN9sc&list=PLRqWX-V7Uu6bJM3VgzjNV5YxVxUwzALHV&index=4>

CodeMonkey Unity graph tutorial: <https://www.youtube.com/watch?v=CmU5-v-v1Qo>

Two Minute Papers: <https://www.youtube.com/channel/UCbfYPyITQ-7l4upoX8nvtg>

