

3D Games Programming

Advanced Game Development with
Advanced Artificial Intelligence.

Box Link:

<https://kingston.box.com/s/iqc3153n0t0zhpi8pl7a7n2g97kfzuy5>

Gordon Johnson

K1451760

Table of Contents

Overview / Links.....	2
Introduction.....	3
Plot	3
Tasks to be Implemented	4
Instructions.....	5
Interfaces	5
Controls	7
Techniques and Algorithms	8
Neural Network.....	8
Genetic Algorithm.....	9
Implemented Source Code.....	11
Neural Network Class	11
NNLayer Class	13
Genetic Algorithm Class	14
Class Diagram of the AI Classes.....	18
Performance and Optimization Techniques	19
Future Improvements and Known Issues.....	20
AI Improvements.....	20
Game Improvements	20
Known issues.....	20
BONUS CONTENT	21
References.....	22
Implementation References.....	22
Asset References	22
Appendix One - GitHub Commits	23

Overview / Links

Teach the machine to play, then play against your creation.

Name: Project Human v Machine

Type: Racing Simulation

Theme: Non-realistic, Racing.

Audience: 3+, Aimed at all age groups

Technology: Unity3D 2017.2

The Box Link: <https://kingston.box.com/s/iqc3153n0t0zhpi8pl7a7n2g97kfzuy5>

GitHub: <https://github.com/LordGee/MachineLearningRaceSim>

PC Version: http://mychaos.co.uk/game/project_hvm/project_hvm.zip

Video: <https://youtu.be/W9QjRx2t2gM>



Figure 1 - Screenshot of actual gameplay

Introduction

The purpose of this game will be to provide a working/practical demonstration, of how Machine Learning can be used to provide an Advance Artificial Intelligence, within an actual game. To develop the Artificial Intelligence, a Neural Network hybrid with a Genetic Algorithm, will be used to train the AI Agent. These agents, depending on performance, can then be used to populate the decision-making process of the AI, that operates on the Neural Network. Once the AI has been trained, the user will have the ability to race against the AI and see who can get the fastest lap time.

Plot

Can you beat the Artificial Intelligence? Allow the machine to train on the tracks, the more time it spends training the better the AI will get and so the more advanced the competition. Offering various game modes, you can choose to train your own Artificial Intelligence in Machine Learning mode, play single player in Human Learning mode, or battle against the previously created AI in Battle mode. See if you can get beat the AI and get the fastest lap time!

Tasks to be Implemented

The tasks to be implemented in this project, have been separated based on the MoSCoW method. This will form a working document and is likely to change throughout the development process.

Must Have:

- Construct one race track.
- Implement one race car, with control mechanics that can be utilised by the human player (Single Player Mode).
- Implement a Neural Network and Genetic Algorithm, to allow Machine Learning.
- Implement appropriate input values, such as ray-casts, to be added into the Neural Network.
- Convert output values, to actual control methods, to enable movement.
- Be able to save and retrieve, trained AI agents.

Should Have:

- Add a verses mode, to allow a human player to battle against the Machine taught AI.
- Event triggers, to improve performance, instead of relying on updates.
- Construct a second race track.
- Create GUI canvas, to provide information feedback to the user, regarding training.
- Create an Options Menu, user interface.

Could Have:

- Audio, Sound Effects and Background Music.
- Construct a third race track.
- Create a GUI interface, to provide lap timer information to the human player.

Won't Have:

- Particle effects.
- Animation.

Instructions

Interfaces

When starting 'Project HvM', the user will be presented with an Option Menu screen. The user can choose from three tracks. Once a track has been selected, the game modes and information will update for that specific track. For example, if the track selected does not have a trained AI Agent, then the Battle Game Mode will disappear. The information below the game modes, will also update displaying the AI Agents current best fitness and the human players best lap time, for the selected track.

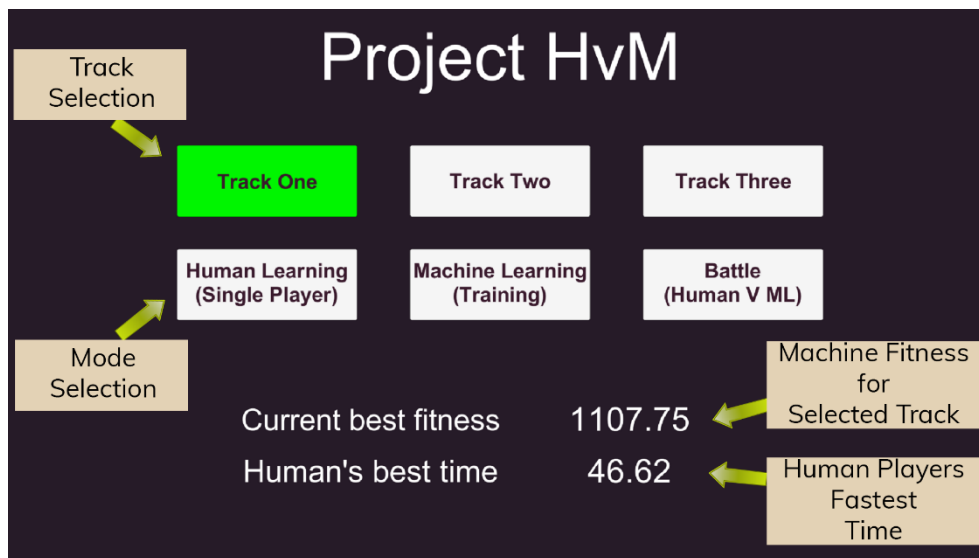


Figure 2 - Main Menu

Once the desired track has been determined, selecting a game mode will launch the appropriate game mode and settings.

For Machine Learning Mode, a specific graphical user interface, is presented to provide the user feedback on current information and performance statistics with regards to the learning process. This information includes the generation, the Machine Learning process is currently on. Once each genome within the current generation is completed, the test results are displayed in the User Interface. While the car is in motion, the fitness level will increase, this is represented in the Current Fitness indicator. The best fitness is the Machines best ever fitness level, this will include previously trained agents.

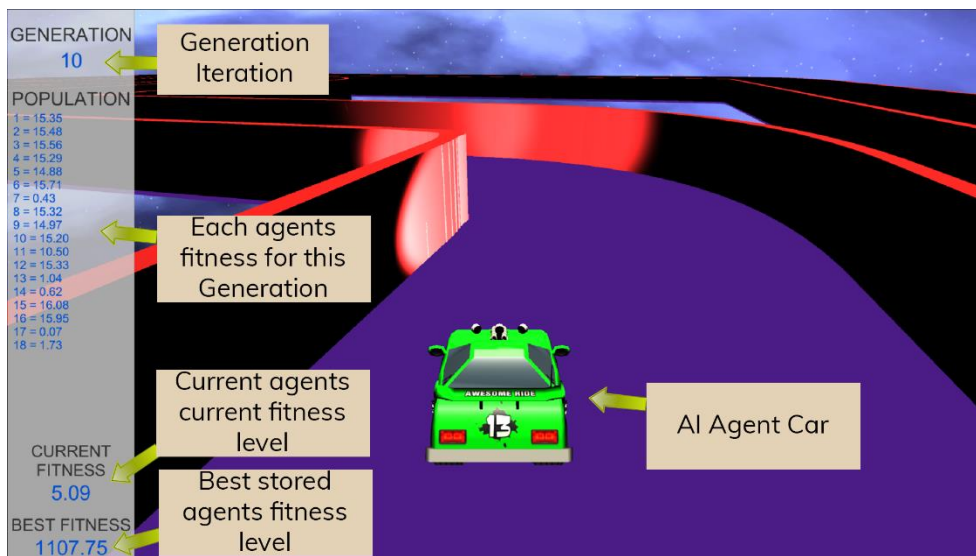


Figure 3 - Machine Learning Interface

For any game mode the human participates in, a graphical user interface displays the appropriate information. This includes the human and machine players best lap times, as well as the current lap timer. The human player car, will always be the blue car.

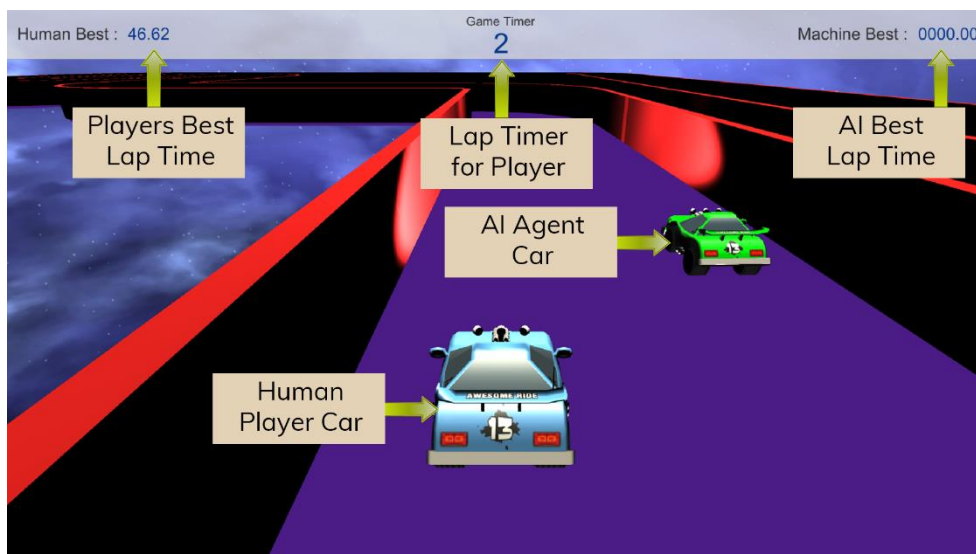


Figure 4 - Player Participation Interface

Controls

This game focuses solely on the keyboard method, for the player to operate this game. However, a gamepad controller should also be available for play.

Keyboard Controls

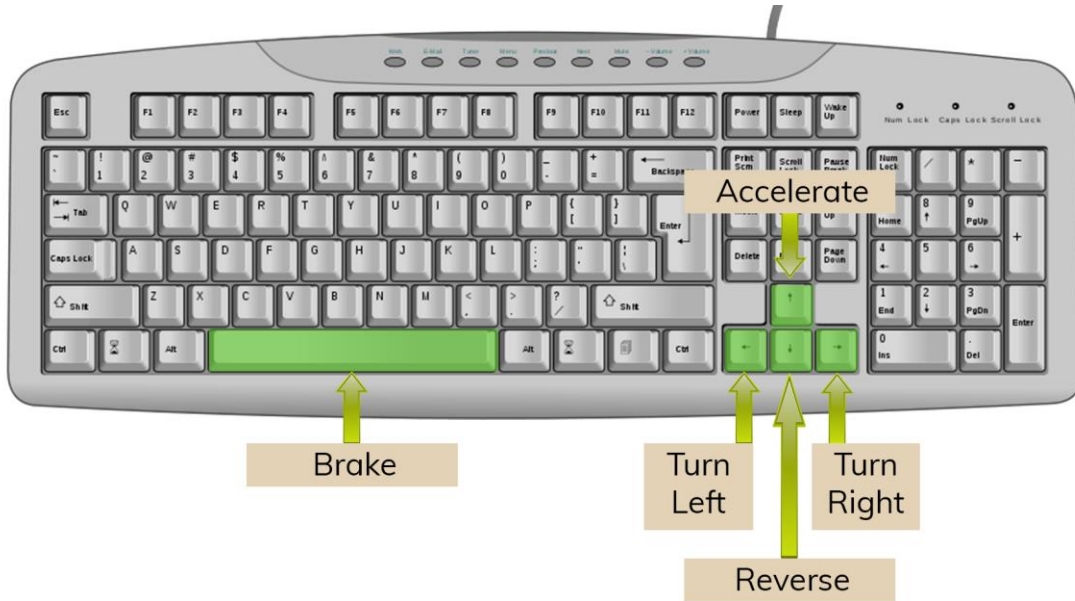


Figure 5 – Player Keyboard Controls



Figure 6 - Machine Learning Keyboard Controls

In addition to the above keyboard layouts, the Escape key, will return the user from the game to the Menu screen. If pressed while in the Menu screen, the application would quit.

Techniques and Algorithms

Neural Network

An Artificial Neural Network (NN) was implemented into this project, for the purpose of allowing the AI Agent to make decisions based on its available data. A NN, consists of several layers, with each layer consisting of many neurons. The first layer, is the input layer. This consists of all the values that are passed through from the sensors, which helps build a picture of the AI Agents surroundings. As seen in figure 7, the car has seven ray-cast inputs, each one populates a neuron within the input layer of the NN, relaying the distance value between the car and the wall barrier. The eightieth input, relays the cars current speed.

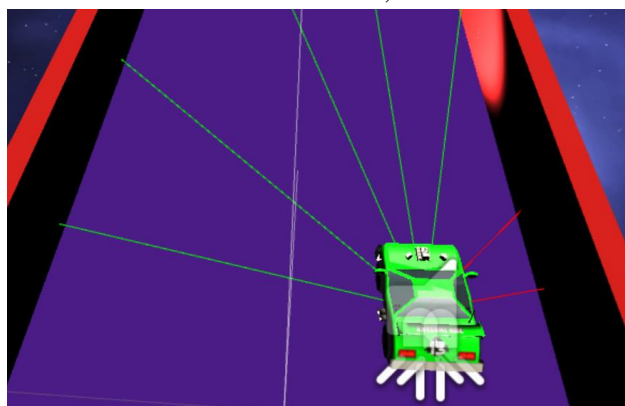


Figure 7 - Ray-cast inputs

The next layer is the hidden layer. This can consist of many layers, there can be many neurons per layer. For this project, the hidden layer has one layer which consists of eight neurons.

The final layer is the output layer. This provides the final values, which then get feed into the controls to drive the car. This layer has four neurons, one for each of the desired outputs. The

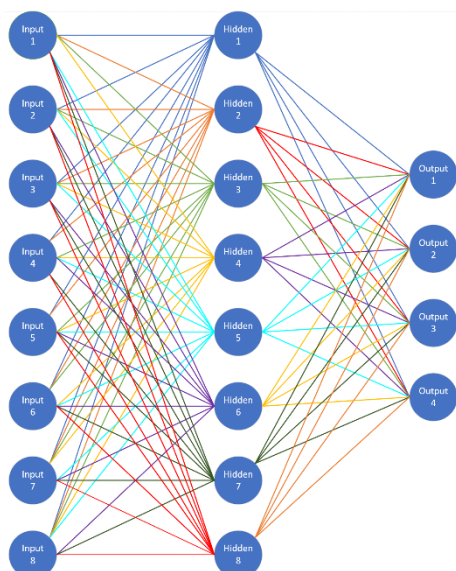


Figure 8 - Neural Network Layout

first two neurons contribute to the steering (horizontal). However, as the project progressed it was apparent that these two neurons could be consolidated into one. The third output neuron, provides the acceleration or reverse value. The final neuron, controls the brake function of the car. This is converted into a Boolean value, at a high threshold, to prevent constant braking.

This type of NN is a feed forward architecture. This means the value will always move in the same direction. Starting at the input, then the hidden layer and finally arriving at the output layer, there is no loop back.

Each connection between two neurons, has a unique weight attached to it. Each incoming connection value, is then evaluated against this unique weight. Each input value, is then multiplied by the weight for that connection. This calculation is accumulated for each connection coming into this neuron, after which an additional weight is accumulated that is multiplied by a bias value, in this example it is (-1). The net value of this, forms the basis of the activation function, this uses the Sigmoid or S-Shaped function to produce a final output value for this neuron, this can then be used in the next layer, if there is one. Alternatives to the Sigmoid function, for this type of project are the step and hyperbolic tangent functions. This process only happens on the Hidden and Output layers. As previously discussed, the input layer receives its initial values from the sensors provided within the game.

To know what weights, should be held in the connection between all these neurons, we need to train this NN to develop appropriate values based on performance. A common method for training a NN is Back-Propagation, learning is achieved by receiving feedback on errors. Although at the early stage of this project, I could not visualise what an error may look like in order to be able to provide that feedback. Due to this, I opted to implement a Genetic Algorithm which trains the NN based on performance and merit, with the best selection of information being passed onto the next generation.

Genetic Algorithm

The Genetic Algorithm (GA), is a process that evolves over time. Each attempt is translated into a fitness value, with the best fitness Genomes being used to generate the next generation.

At the beginning, the GA will create a defined population or generation of Genomes. For this project, the number Genomes was set to 20. This value should typically be an even, allowing for cross breeding later. Each Genome, is created by generating a set of random weights, ranging between -1.0 and +1.0, this process is repeated for all the possible connections in the NN. This is formulated by: Input Neurons * Hidden Neurons * Output Neurons + Hidden Neurons + Output Neurons.

For this project: $8 * 8 * 4 + 8 + 4 = 268$ weights per Genome or generation.

Once this first generation has been instantiated, each Genome is applied to the NN for testing. While the Genome is being tested, its current fitness level is recorded, based on speed of the car. The further the Genome travels, the higher the fitness level and so, the better the performance. Once testing has exhausted all Genomes, then a breeding process takes place, to develop the next generation of Genomes.

The breeding process, selects the best cases from the previous generation, in this example that is the best four Genomes. Each of these four Genome weights, are copied into the next population, where each is passed through a mutation process that has an 8% chance of mutating a given weight (more on mutation later). As an extra condition, I added the ability to store and reproduce the best performing genome into the next generation, un-mutated. This was an attempt to prevent the downgrading of abilities from one generation to the next.

After these Genomes have been added, a cross breeding process begins. Taking two of the best Genomes at a time and generating two child Genomes out of these. The first is taken, with a random amount of weight taken and used for the start of one of the child's weights, the remainder is then made up of the second Genome. This child, then goes through the mutation process, which provides the chance for these be altered again. Alternatively, there are other methods that can be explored here in later projects, such as taking every other one in a crisscross fashion, or assigning each weight from a completely random Genome.

Example of cross breeding:

Genome 1	Genome 2	Child 1
0.8	-0.2	0.8
0.6	-1	0.6
-0.5	0.3	-0.5
-0.2	0.8	-0.2
0.0	0.6	0.6
-0.9	-0.6	-0.6
1.0	-0.7	-0.7

Finally, if the Genomes that have been generated, do not meet the maximum population, completely brand-new Genomes with random weight are generated and added until the list holds the right population value.

The new generation, is then put to the test and the process continues.

The Mutation process, provides an 8% chance that each weight could potentially change. there are four different methods that could affect weight change. For example, one of these changes could mean the weight value goes from a positive number to a negative, or another could be that a completely new value is randomly generated. Based on the amount of weights held in this configuration, about 21 different weights could be mutated on average.

Implemented Source Code

Before implementation, I referred to many sources of information. Including various books (Bourg and Seemann, 2004), (Buckland, 2005), (Mueller and Massaron, 2016), (Schwab, 2009). As well as some online resources, that provided a coding structure in C++ (Robbins, 2014), which helped organise my classes and provided a pseudocode structure for the functions.

Neural Network Class

The neural network class, is the centre of the artificial intelligence. There are two main functions within the class, that contribute to this workload; the Update NN and the Populate Neurons from Genome functions.

Update NN Function

```
/// <summary>
/// This function handles the updating of the Neural Network.
/// 1. Creates an empty list for the outputs for each layer
/// 2. Starts by iterating through the hidden layer(s) then
/// the output layer
/// 3. If its the first iteration that the inputs are
/// provided by the input manager e.g. from the sensors.
/// 4. For all other iteration the outputs from the previous layer
/// become the inputs.
/// </summary>
public void UpdateNN() {
    outputs = new List<float>();
    for (int i = 0; i < hiddenLayers.Count; i++) {
        if (i > 0) {
            inputs = outputs;
        }
        hiddenLayers[i].Evaluate(inputs, ref outputs);
    }
    inputs = outputs;
    outputLayer.Evaluate(inputs, ref outputs);
}
```

While the Update NN Function, is a simple function, it's purpose is to call an evaluation of the hidden and output layers on every frame.

Populate Neurons from Genome Function

```

/// <summary>
/// This class populates each neuron form the weight provided from the Genome.
/// Generating a new Neural Network, this provides the infrastructure for the
/// decision making process.
/// </summary>
/// <param name="_genome">Current genome to be executed</param>
/// <param name="_input">Total number of inputs</param>
/// <param name="_neuronsPerHidden">Total number of neurons per hidden
layer</param>
/// <param name="_output">Total number of outputs</param>
public void PopulateNeuronsFromGenome(ref Genome _genome, int _input, int
_neuronsPerHidden, int _output) {
    ReleaseNN();
    outputAmount = _output;
    inputAmount = _input;
    NNLayer hidden = new NNLayer();
    List<Neuron> hiddenNeurons = new List<Neuron>();
    hiddenNeurons.Capacity = _neuronsPerHidden;
    for (int i = 0; i < _neuronsPerHidden; i++) {
        List<float> weights = new List<float>();
        weights.Capacity = _input + 1;
        for (int j = 0; j < _input + 1; j++) {
            weights.Add(_genome.weights[i * _neuronsPerHidden + j]);
        }
        hiddenNeurons.Add(null);
        hiddenNeurons[i] = new Neuron();
        hiddenNeurons[i].weights = weights;
        hiddenNeurons[i].numberOfInputs = _input;
    }
    hidden.LoadLayer(hiddenNeurons);
    hiddenLayers.Add(hidden);
    List<Neuron> outputNeurons = new List<Neuron>();
    outputNeurons.Capacity = _output;
    for (int i = 0; i < _output; i++) {
        List<float> weights = new List<float>();
        weights.Capacity = _neuronsPerHidden + 1;
        for (int j = 0; j < _neuronsPerHidden + 1; j++) {
            weights.Add(_genome.weights[i * _neuronsPerHidden + j]);
        }
        outputNeurons.Add(null);
        outputNeurons[i] = new Neuron();
        outputNeurons[i].weights = weights;
        outputNeurons[i].numberOfInputs = _input;
    }
    outputLayer = new NNLayer();
    outputLayer.LoadLayer(outputNeurons);
}
}

```

The Populate Neurons from Genome function, populates the hidden and output layers neuron connections, with the weights supplied from the next active Genome. This prepares the NN, for the upcoming test attempt.

NNLayer Class

The NNLayer class is the basis of each layer. The important function here is the Evaluation function, which executes the Activation process for Neurons within the given layer.

Evaluate Function

```

    /// <summary>
    /// This function evaluates the inputs of the previous layer, by iterating through
    each neuron
    /// performing a calculation that will be a primer for the output value which is
    accumulated by
    /// (input value * each weight of a neuron). An additional value is also added is
    an additional
    /// neuron weight that is multiplied by the bias which is -1.0f. this is then
    added to the list
    /// of outputs after performing a Sigmoid calculation.
    /// </summary>
    /// <param name="_input">List of input values (output from previous layer if not
    the first)</param>
    /// <param name="_output">Output list result after evaluating the results</param>
    public void Evaluate(List<float> _input, ref List<float> _output) {
        int inputIndex = 0;
        for (int i = 0; i < totalNeurons; i++) {
            float activation = 0.0f;
            for (int j = 0; j < neurons[i].numberOfInputs - 1; j++) {
                activation += _input[inputIndex] * neurons[i].weights[j];
                inputIndex++;
            }
            activation += neurons[i].weights[neurons[i].numberOfInputs] *
            ConstantManager.BIAS;
            _output.Add(Sigmoid(activation, 1.0f));
            inputIndex = 0;
        }
    }

```

This function was explained in the previous section of this report, regarding layer evaluation, evaluation, activation function and the Sigmoid function, (see page 8).

Genetic Algorithm Class

The genetic algorithm, manages the creation of new Genomes and the breeding process for new generations.

Create New Genome Function

```

    /// <summary>
    /// Generates a new genome and populates starting weight float values
    /// between -1.0 and +1.0. Typically, only used at the start
    /// </summary>
    /// <param name="_totalWeights">Total weights required for this
configuration</param>
    /// <returns></returns>
    private Genome CreateNewGenome(int _totalWeights) {
        Genome genome = new Genome();
        genome.ID = genomeID;
        genome.fitness = 0.0f;
        genome.weights.Capacity = _totalWeights;
        for (int i = 0; i < _totalWeights; i++) {
            genome.weights.Add(Random.Range(-1.0f, 1.0f));
        }
        genomeID++;
        return genome;
    }

```

Generates a brand-new Genome with its own unique weights, to the total amount required for the set configuration.

Generate New Population Function

```

    /// <summary>
    /// Populates a brand-new population of genomes up to the
    /// maximum population total.
    /// </summary>
    /// <param name="_newTotalPopulation">Maximum population for all
generations</param>
    /// <param name="_totalWeights">Total weights required for this
configuration</param>
    public void GenerateNewPopulation(int _newTotalPopulation, int _totalWeights) {
        generation = 1;
        ClearPopulation();
        currentGenome = 0;
        totalPopulation = _newTotalPopulation;
        population.Capacity = _newTotalPopulation;
        for (int i = 0; i < population.Capacity; i++) {
            Genome genome = CreateNewGenome(_totalWeights);
            population.Add(genome);
        }
        EventManager.TriggerEvent(ConstantManager.UI_GENERATION, generation);
    }

```

Depending on the set maximum population, this function generates the desired number of Genomes for the first generation.

Breed Population Function

```

/// <summary>
/// This function is called when the previous generation has completed
/// all tests. The four best genomes from the previous generation are identified
/// The four best are then added to the next generation after the mutation process
/// has been complete. It then adds the overall best performing genome which does
/// not receive mutation. After this a cross breeding process takes place, each
/// child that is returned is then also mutated. Lastly if there are any remaining
/// spaces to reach maximum population then brand new random genomes are generated
/// and added to the next population.
/// </summary>
public void BreedPopulation()
{
    List<Genome> bestGenomes = new List<Genome>();
    GetBestCases(ConstantManager.NUMBER_OF_GENOMES_TO_BREED, ref bestGenomes);
    List<Genome> children = new List<Genome>();
    Genome topGenome = new Genome();
    for (int i = 0; i < bestGenomes.Count; i++) {
        SetUpTopGenome(ref topGenome, bestGenomes[i]);
        Mutate(topGenome);
        children.Add(topGenome);
    }
    SetUpTopGenome(ref topGenome, bestEverGenome);
    children.Add(topGenome);
    Genome child1 = null;
    Genome child2 = null;
    int crossBreedIteration = Mathf.Abs((ConstantManager.MAXIMUM_GENOME_POPULATION
- children.Count) / (bestGenomes.Count + 2));
    for (int i = 0; i < crossBreedIteration; i++) {
        for (int j = 1; j < bestGenomes.Count; j++) {
            CrossBreed(bestGenomes[i], bestGenomes[j], ref child1, ref child2);
            Mutate(child1);
            children.Add(child1);
            Mutate(child2);
            children.Add(child2);
        }
    }
    int remainingChildren = totalPopulation - children.Count;
    for (int i = 0; i < remainingChildren; i++) {
        children.Add(CreateNewGenome(bestGenomes[0].weights.Count));
    }
    ClearPopulation();
    population = children;
    currentGenome = 0;
    generation++;
    EventManager.TriggerEvent(ConstantManager.UI_GENERATION, generation);
}

```

This function, is described in the previous section of this report, (see page 9).

Cross Breed Function

```

/// <summary>
/// This function takes in two existing Genomes and takes the weights and
/// splits them at a random section and populates the remaining weights with
/// the second genome to ensure the Capacity remains unchanged, this produces
/// two babies which have a mixer of the two-original genome.
/// </summary>
/// <param name="_g1">Original genome one</param>
/// <param name="_g2">Original genome two</param>
/// <param name="_baby1">Returned baby one</param>
/// <param name="_baby2">Returned baby two</param>
private void CrossBreed(Genome _g1, Genome _g2, ref Genome _baby1, ref Genome
_baby2) {
    int totalWeights = _g1.weights.Capacity;
    int crossOver = Random.Range(0, totalWeights);
    _baby1 = new Genome();
    _baby1.ID = genomeID;
    _baby1.weights.Capacity = totalWeights;
    genomeID++;
    _baby2 = new Genome();
    _baby2.ID = genomeID;
    _baby2.weights.Capacity = totalWeights;
    genomeID++;
    for (int i = 0; i < crossOver; i++) {
        _baby1.weights.Add(_g1.weights[i]);
        _baby2.weights.Add(_g2.weights[i]);
    }
    for (int i = crossOver; i < totalWeights; i++) {
        _baby1.weights.Add(_g2.weights[i]);
        _baby2.weights.Add(_g1.weights[i]);
    }
}

```

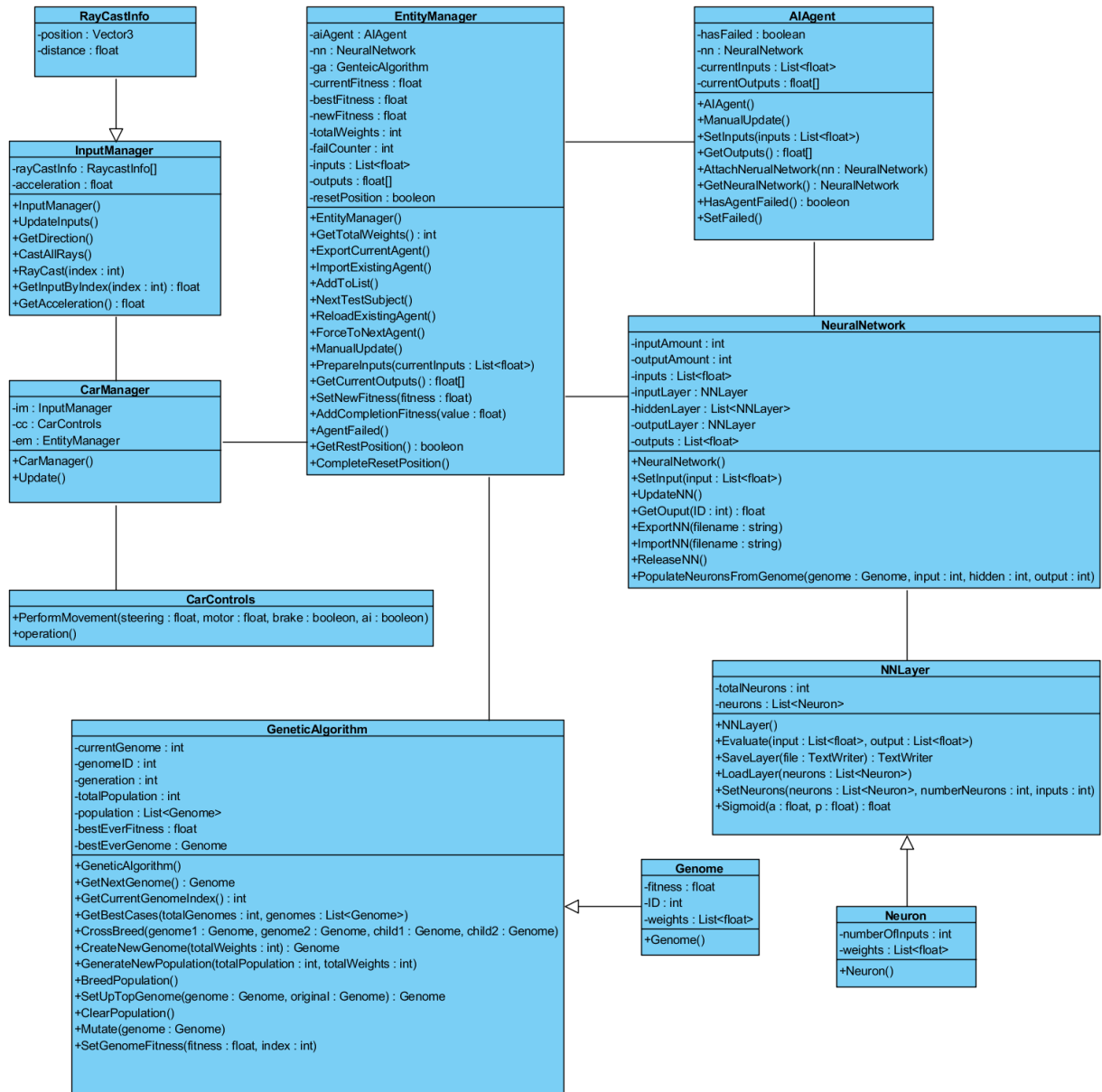
This function is described in the previous section of this report, (see page 10).

Mutate Function

```
/// <summary>
/// The class provides an 8% mutation rate for a genome that is
/// passed into it. Depending on the random hit, provides different
/// effects on the current weight value.
/// </summary>
/// <param name="_genome">Genome that requires to be mutated</param>
private void Mutate(Genome _genome) {
    for (int i = 0; i < _genome.weights.Count; i++) {
        float mutationLottery = Random.Range(0f, 100f);
        if (mutationLottery <= 2f) {
            _genome.weights[i] *= -1;
        } else if (mutationLottery <= 4f) {
            _genome.weights[i] = Random.Range(-0.5f, 0.5f);
        } else if (mutationLottery <= 6f) {
            _genome.weights[i] *= Random.Range(1.0f, 2.0f);
        } else if (mutationLottery <= 8f) {
            _genome.weights[i] *= Random.Range(0.0f, 1.0f);
        }
    }
}
```

This function is described in the previous section of this report, (see page 10).

Class Diagram of the AI Classes



Performance and Optimization Techniques

When considering performance and optimisation techniques, I considered some of the advice given by our recent guest industry speaker, Haris Kapagioridis. One of the main points that came across to improve performance, is to minimise the amount of Destroying and Instantiating methods, used within the game.

Initially when first implementing this project, the player car was Instantiated at the beginning of the scene. On colliding with a barrier, it was then destroyed and re-instantiated at the Spawn position. This is no longer the case, now the car resets its position and statistics back to its starting values, meaning that there is no need to Instantiate this object again, until another track or game mode is started.

Another notable point, was to use an Event system to effect changes in other classes, without having to check whether an update is needed on every frame. I implemented this half way through the project when developing the User Interface (UI). This become increasing useful for displaying a Genome performance to the UI. This only needed to be updated as and when the Genome had finished its test. Upon completion of the Genomes task, I was able to trigger an event that would signal to the UI class, to update the display with the latest information. However, this still required the UI class to back track through the classes, in order to retrieve the information to perform the update. To improve this, I did a bit of investigation into allowing values such as a float to be passed through when triggering an event. This allowed the event to trigger the correct function and relay the current information needed to be populated, in one line of code.

Using the official (Unity3D, 2017) tutorial for creating a simple messaging system and referring to a forum post (Unity Forums, 2015) for passing through a variable, I was able to create an event manager with overloading, to allow floats to be passed or no variable to be passed. This obviously can be modified for all data types and combinations, depending on the requirements of the application. In hindsight it would have been extremely useful to have developed this system earlier in the project, as I can see how this could have benefited and even improved the performance of the AI classes.

Another improvement that was made, was the introduction of a constant manager. While this does not improve the performance of the application, it improves workflow, saving time where string names need to be exact from one class to the next, as well as providing a central location for constant values, that are reused throughout the application.

Future Improvements and Known Issues

AI Improvements

- Make improvements to the breeding process in the genetic algorithm class, to further reduce the risk of degrading performance.
- Improve the cross-breeding process, by exploring alternatives to splitting the weights ratio, rather than using the start and end of a Genome.
- Add the ability to store and reload a complete generation of Genomes, so training can continue from where it left off. Currently training will always start with random values.
- Improve the fitness reward, to incorporate the time in which the lap was completed. This would provide an additional incentive, for the AI to progress the track faster, rather than slower.
- Implement event system into the AI classes.

Game Improvements

- Implement animation and effects to improve the game cosmetics.
- Build track pieces in a proper 3D modelling tool. The current method of track pieces made with a Bezier Spline, has affected performance.
- Improve the directional lighting, to effectively light up the track.
- Add another game mode, to include multiple AI agents at the same time.
- Allow the choosing of a specific Genome in which to race, currently only the best performing Genome is selected to be loaded back in, although others are stored.

Known issues

- (Build only version) AI Agent, when learning will sometimes hang with no movement whilst still generating an excessive fitness level.
- Track Two supplied AI, does not always complete the lap (50% of the time).

BONUS CONTENT

At the end of the implementation, I purposely left 'Track Three' untrained, to demonstrate how the User Interface will differ when an Agent for a Track has not been trained yet. However, if you would rather not train a new agent, the game does include a trained agent file to race this track, simply add the following two lines to the Assets/Data/List.csv.

```
3  
1128.085
```

Once saved, the Human vs Machine button will become available. Allowing you to race against the Machine in Track Three, without further training.

References

Implementation References


- Bourg, D. and Seemann, G. (2004) *AI for Game Developers*. United States: O'Reilly.
- Buckland, M. (2005) *Programming Game AI by Example*. Texas: Wordware Publishing.
- Catlike Coding (2017) *Curves and Splines, making your own path*. Available at: <http://catlikecoding.com/unity/tutorials/curves-and-splines/> (Accessed: 12 November 2017).
- Mueller, J. and Massaron, L. (2016) *Machine Learning for Dummies*. New Jersey: John Wiley & Sons.
- Robbins, M. (2010) *Neural Network Demo*. Available at: <https://www.youtube.com/watch?v=0Str0Rdkxxo> (Accessed: 10 November 2017).
- Robbins, M. (2014) *Neural-Network*. Available at: <https://github.com/matthewrdev/Neural-Network> (Accessed: 10 November 2017).
- Schwab, B. (2009) *AI Game Engine Programming*. Boston: Course Technology.
- Unity3D (2017) *Events: Creating a simple messaging system*. Available at: <https://unity3d.com/learn/tutorials/topics/scripting/events-creating-simple-messaging-system> (Accessed: 26 November 2017).
- Unity3D (2017) *Layers*. Available at: <https://docs.unity3d.com/Manual/Layers.html> Available at: <https://docs.unity3d.com/Manual/Layers.html> (Accessed 19 November 2017).
- Unity Forums (2015) *[Messaging System] Passing parameters with the event*. Available at: <https://forum.unity.com/threads/messaging-system-passing-parameters-with-the-event.331284/> (Accessed: 26 November 2017).

Asset References


- Debsound (2015) *Rally Car Idle Loop 17.wav*. Available at: <https://freesound.org/people/debsound/sounds/278186/> (Accessed: 06 December 2017)
- Hedgehog Team (2012) *Skybox Volume 2 (Nebula)*. Available at: <http://u3d.as/2We> (Accessed: 15 November 2017).
- Reitanna (2016) *big thud2.wav*. Available at: <https://freesound.org/people/Reitanna/sounds/332668/> (Accessed: 07 December 2017)
- Romariogrande (2016) *Space Chase*. Available at: <https://freesound.org/people/Romariogrande/sounds/370801/> (Accessed: 06 December 2017)
- UltimateArcade (2014) *Toon Race Car - Low Poly*. Available at: <http://u3d.as/84T> (Accessed: 12 November 2017)


Appendix One - GitHub Commits

Commits on Nov 10, 2017


 **Road Surface** ...
LordGee committed on 10 Nov
Road Surface: <https://pixabay.com/en/seamless-repeat-repetitive-2033661/>
Arrow: <https://pixabay.com/en/arrow-bright-abstract-red-698691/>


TODO: Create Spline, read this tutorial:
<http://catlikecoding.com/unity/tutorials/curves-and-splines/>


 **Initial Project Setup**
LordGee committed on 10 Nov

 **Initial commit**
LordGee committed on 10 Nov

Commits on Nov 12, 2017

 **Trying to get the car to move**
LordGee committed 28 days ago

 **Minimised Objects in Scene** ...
LordGee committed 28 days ago
+ Manually removed all un-needed track peices from the scene. (3000 down to 480)
improved performance and still have a track.

 **Track Builder** ...
LordGee committed 28 days ago
+ Followed a tutorial to use a Bezier Spline to generate a race track.
ref: <http://catlikecoding.com/unity/tutorials/curves-and-splines/>
+ Added a car to the scene
+ Presently adding control features

Commits on Nov 15, 2017



Successfully navigate the Track 1 ...

LordGee committed 25 days ago

- + Added start and end game to the scripts
- + Revisited the control mechanic to minimise unneeded functionality
- + Added a car manager script the deals with collisions
- + Added a game controller which manages the main game variables, respawns and end games stats.
- + Added a skybox ref <http://u3d.as/2We>



Car now working properly ...

LordGee committed 25 days ago

- + Added new track (little simpler)
- + Sorted out some issues with the wheel collider settings

Commits on Nov 17, 2017



First attempt at a Neural Network ...

LordGee committed 24 days ago

- + This will probably be overwritten by the end of today

Commits on Nov 18, 2017



Second Attempt

LordGee committed 22 days ago

Commits on Nov 19, 2017



RayCasts and NN and Genetic Algo ...

LordGee committed 21 days ago

- + Car now projects three raycasts ~ current issue, when locating gameobject by tag, returning 8 objects instead of three.
- + Third attempt with the neural network, now incorporating Genetic Algorithm. To be tested.

Commits on Nov 20, 2017



Constant Manager + Minor Improvements ...

LordGee committed 20 days ago

- + Added constant manager so all the constant variables are in one location, could be helpful if kept organised.
- + Added an agent scripts which will attach to the car and make the connection with the Neural Network.
- + Added entity manager script

Commits on Nov 22, 2017



Improvements made to the EntityManager script

LordGee committed 18 days ago



Minor progress ...

LordGee committed 18 days ago

- to the
 - Agent, EntityManager and ConstantManager scripts

Commits on Nov 23, 2017



Improving Input Support ...

LordGee committed 17 days ago

- + Improving the input collection that gets passed through the NN
- + Refactored the control script to double up that allows for AI control
- + Made improvements to the RayCast manager to be more generic, also now collects other inputs (Will probably change the class name to the Input Manager)

Commits on Nov 24, 2017



Removing errors / inconsistencies

LordGee committed 16 days ago



Fixed Raycast issues ...

LordGee committed 16 days ago

- + Raycasts now point in the forward direction of the car.
- + Raycasts left and right are now spread by a factor of 0.25f
- + Raycasts now ignore the player car.

Commits on Nov 25, 2017



MILE STONE: NN and GA now working

LordGee committed 15 days ago

Commits on Nov 26, 2017



Event Manager + Refactor + UI Canvas

LordGee committed 14 days ago

Commits on Nov 27, 2017



Preventing movement after respawn

LordGee committed 13 days ago

Commits on Nov 28, 2017



Improvements made to GA ...

LordGee committed 13 days ago

- + Modifications made to the genetic algorithm
- + Added ability to change system time scale when training

Commits on Nov 30, 2017



Minor Update ...

LordGee committed 10 days ago

- + Ensures the fitness value stays constant across learning speeds
- + Added some additional input sensors to try and improve accuracy

Commits on Dec 1, 2017



Saving NN to Text file

LordGee committed 9 days ago

Commits on Dec 2, 2017








Importing Data ...

LordGee committed 8 days ago

Import function written, testing need to still take place.




Commits on Dec 3, 2017

-  **Menu Track Options** ...
LordGee committed 7 days ago
Loads correct files for given track
Identifys best fitness to load
Menu is all linked now
-  **Main Menu**
LordGee committed 7 days ago
-  **minor change**
LordGee committed 7 days ago
-  **Successfully loaded in a previous agent**
LordGee committed 7 days ago
-  **Alt Fitness Level** ...
LordGee committed 7 days ago
Ensures fitness levels stay consitant at various game speeds


Commits on Dec 4, 2017


-  **Minor Push**
LordGee committed 6 days ago




Commits on Dec 6, 2017

-  **Refactoring plus NEW TRACK**
LordGee committed 4 days ago
-  **Refactoring and Commenting code**
LordGee committed 4 days ago
-  **Building UI for player mode**
LordGee committed 4 days ago



Commits on Dec 7, 2017

-  **Menu Sound + NEW Track + FINAL COMMIT** ...
LordGee committed 3 days ago
Thud : <https://freesound.org/people/Reitanna/sounds/332668/>


+ Added the third and final track
-  **Sound effects** ...
LordGee committed 3 days ago
Reference:
Engine : <https://freesound.org/people/debsound/sounds/278186/>
Music : <https://freesound.org/people/Romariogrande/sounds/370801/>

Almost done just need to:
+ Add sound for menu
+ Add one more track (untrained)
-  **Training track 2 agents**
LordGee committed 3 days ago
-  **Refactoring and Commenting**
LordGee committed 3 days ago
-  **Ready for Training Track 2**
LordGee committed 4 days ago

Commits on Dec 8, 2017

-  **Added missing skybox** ...
LordGee committed 2 days ago
+ Added skybox to tracks 2 & 3
+ Removed track 3 training records from the list csv file
-  **Training results**
LordGee committed 3 days ago

Commits on Dec 10, 2017

-  **Final Commit + Minor Improvements**
LordGee committed 9 minutes ago