Bahria University

Karachi Campus

**COURSE: Software Construction**

**REPORT**

**SUBMITTED BY:**

**Moeez Ahmed Shah (02-131182-045)**

**SUBMITTED TO:**

Engr.Misbah Parveen

# Project name:

Arcade AI (A 3d game)

**The clowns**

13-D 1, Gulshan-e-Iqbal

Karachi, 80808 (123) 456-7890

# Intended use:

Ideally, what boss wants is for his gaming zone to become famous and the gamers at his game gaming zone to represent it better by being better at games. Currently, as a manager of that gaming zone, he has a strong belief that the gamers at his zone have a lot of potential to represent Pakistan in Exports community globally, but the gamers at his zone aren’t professional enough to compete in such big events as they do not have any kind of good practicing platform in order to enhance their skills and take them to next level. As the gamers cannot represent the country and his gaming zone globally, he is losing customers (gamers) quite frequently as there are some zones that provide many better practicing facilities in order to enhance skills of their gamers. So his gaming zone seeks to resolve this issue by creating some kind of practice game that can help the gamers to enhance their skills so they could represent his gaming zone better at international and national levels and be more competitive.

# Feature/Accessories:

Major feature of this game includes:

MF-1: Player fight with Zombie.

MF-2: Player watch all scene through camera.

MF-3: Can select the level.

MF-4: Game generate Machine Learning Agents.

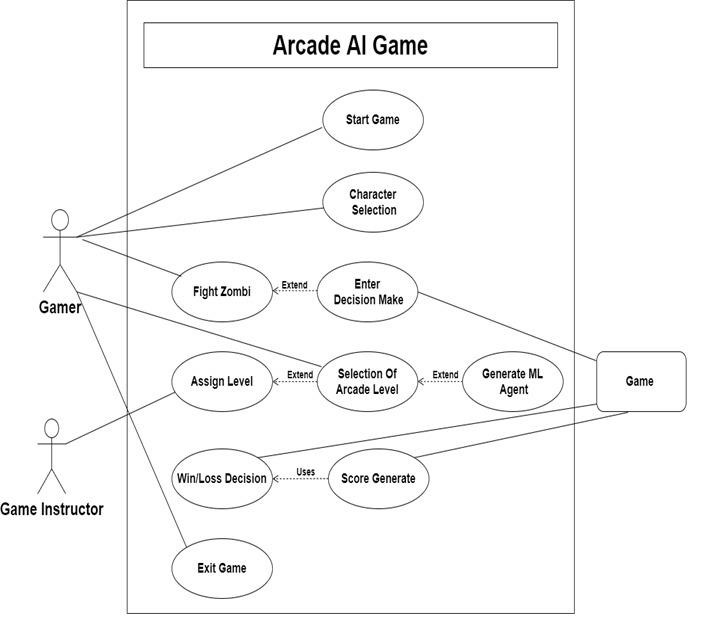
MF-5: Player control all direction.

MF-6: Multiple difficulty portion available.

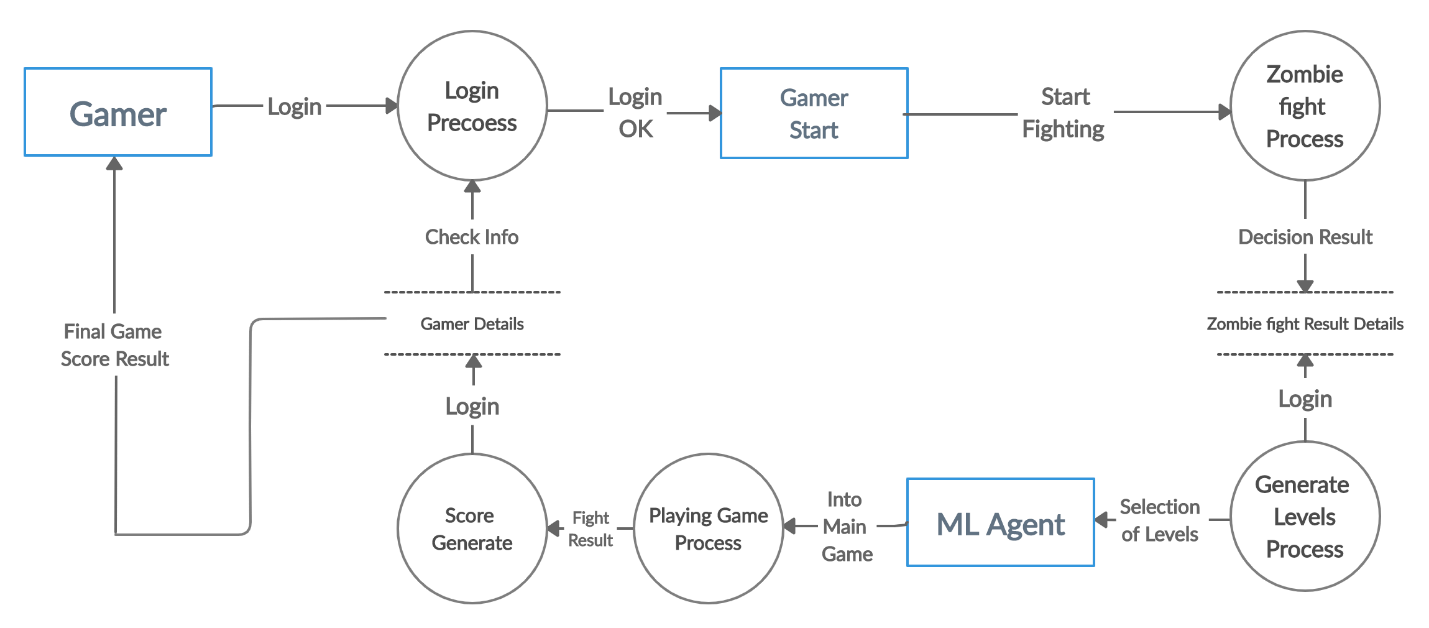
MF-7: Module 2 and 3 provides a fantastic twist for assessment.

MF-8: Getting an assessment based on their gameplay.

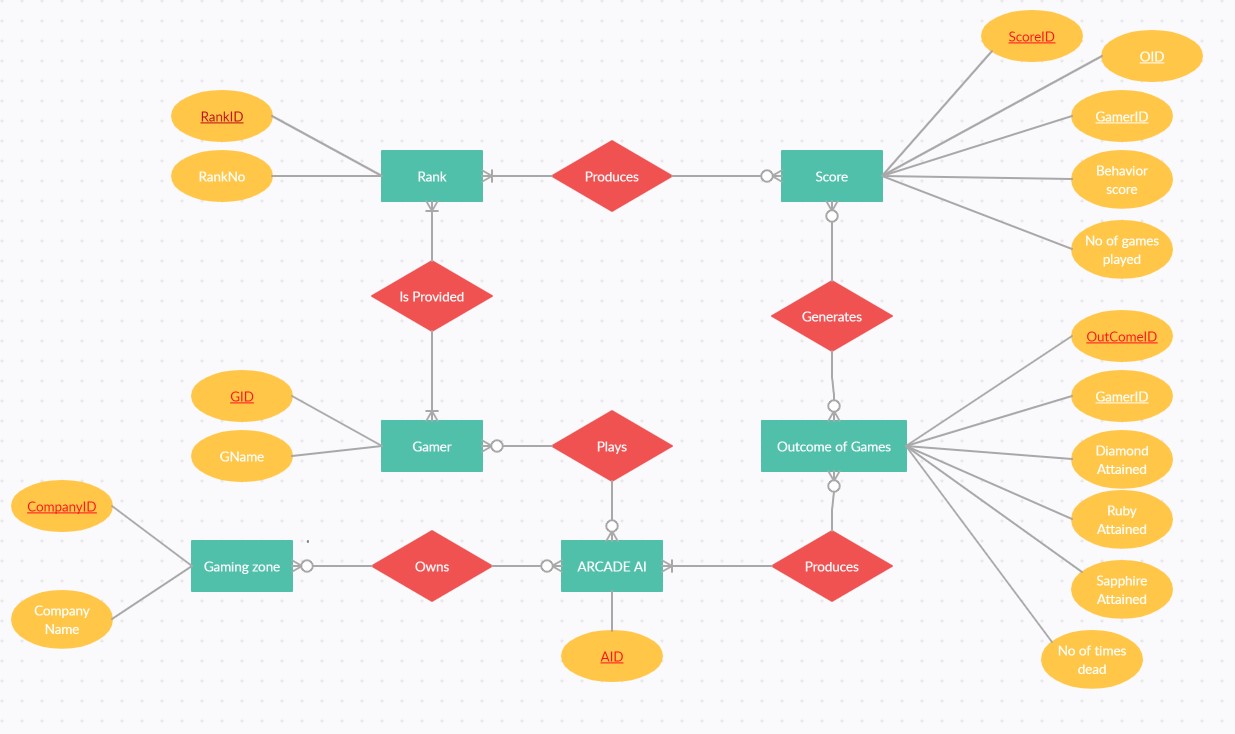
# Description of the main project elements:



**Fig 1:** Use case Diagram

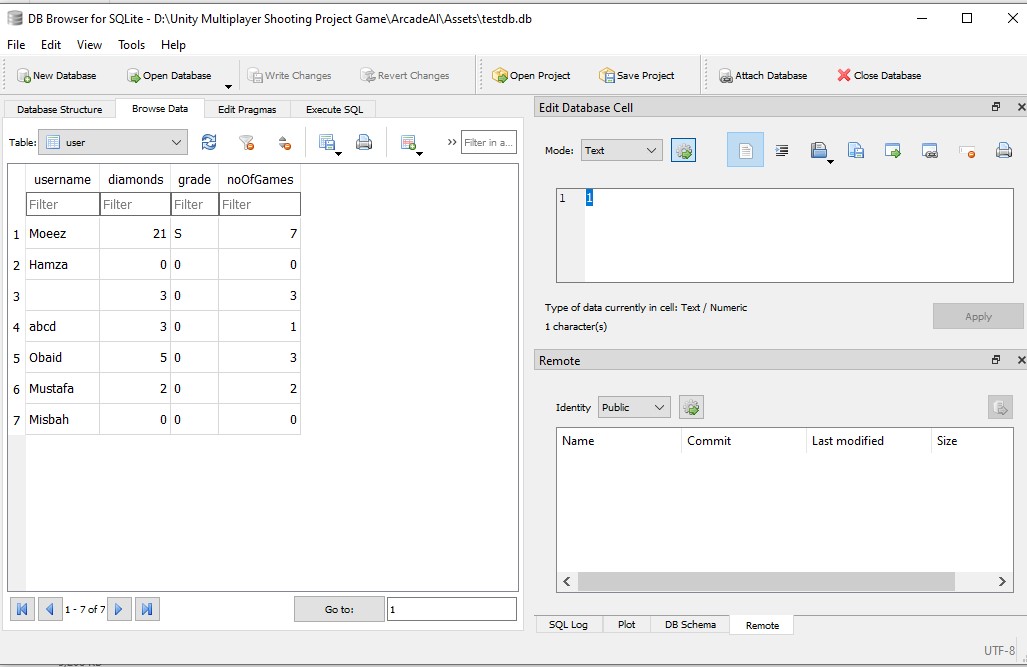


**Fig 2:** Data Flow Diagram of Arcade AI.



**Fig 3:** Entity Relationship Diagram for Arcade AI.

Database = SQLite.



Information on 1st panel shows the profile of player, that is:



Module 1 is our first module. In this particular module the player plays as the chosen one and their job is to find the skeleton boss, kill him. The killing is pretty hard as there is a lot happening in this module. There is also a summon skeleton of the skeleton boss. The skeleton boss has 3 attacks and attack is based on random. The skeleton minion is an AI based bot (NPC) that moves from place to place in order to search for the chosen one and kill him, perhaps. The player (chosen one) would get one diamond for defeating the skeleton boss. That 1 diamond would be a key for entering next modules. There are 2 doors for next modules, chosen one may choose any of them. The diamond also acts as the assessment future, the more the diamond the better the person performed in the game.

Chosen one has 6 attacks. The first attack is an earthquake slam ground, the second spell is a simple hit that deducts 5 health of the target hit. The 3rd attack is like a fairy wind that uses ray cast in order to go in straight line and if it hits a target named enemy then it damages them, that is 30 damage. The 4th spell is the shield for our chosen one, when under this spell, the enemies may not be able to hit us at any cost. The 5th attack is healing that heals 10 – 20 hp randomly of the chosen one. The 6th attack is the ultimate spell that sends thunder in all paths damaging 30-60 health if enemies get hit. Every attack has it`s own unique animation. Every attack has a timer ranging from 3-15 seconds. Health at beginning = 100, and panel is displayed on the left.



Skeleton boss has 3 attacks. 1st attack is simple and 2nd attack is the hit attack, that is same as the chosen one. The 3rd attack puts the chosen one on fire and damages them a lot, for around 20-30 health.

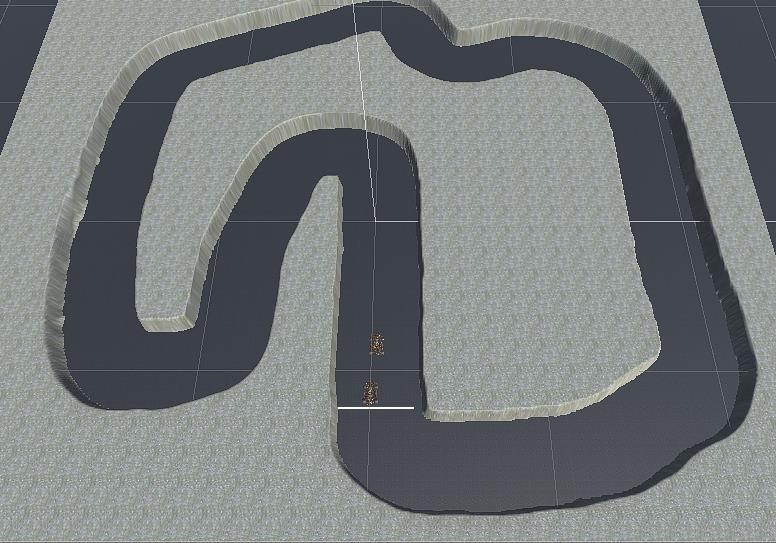
The mouse is customized, the movement is also customized, the terrain is unique to our game and a lot more. The skeleton and skeleton boss uses A\* search in order to find the walkable path and the enemy in it`s range.

Module 2 is our 2nd module and sometimes 3rd, depending on what person choose in the first module. In this module the chosen one has to find a certain girl that is known for her pretty appearance. The crowd uses A\* search algorithm in order to find and pick up their random goal states and decide the walkable path. Crowd also have awareness of avoiding the chosen one in case of collision. If chosen one collides with the target girl then the girl disappears dropping the diamond hence making us eligible for next stage.

For finding the next stage, chosen one needs to find a specific shop named old tradfford. However, if chosen one faces difficulty in order to find it than after 20 seconds the shop name will be displayed in instructions (using coroutines).

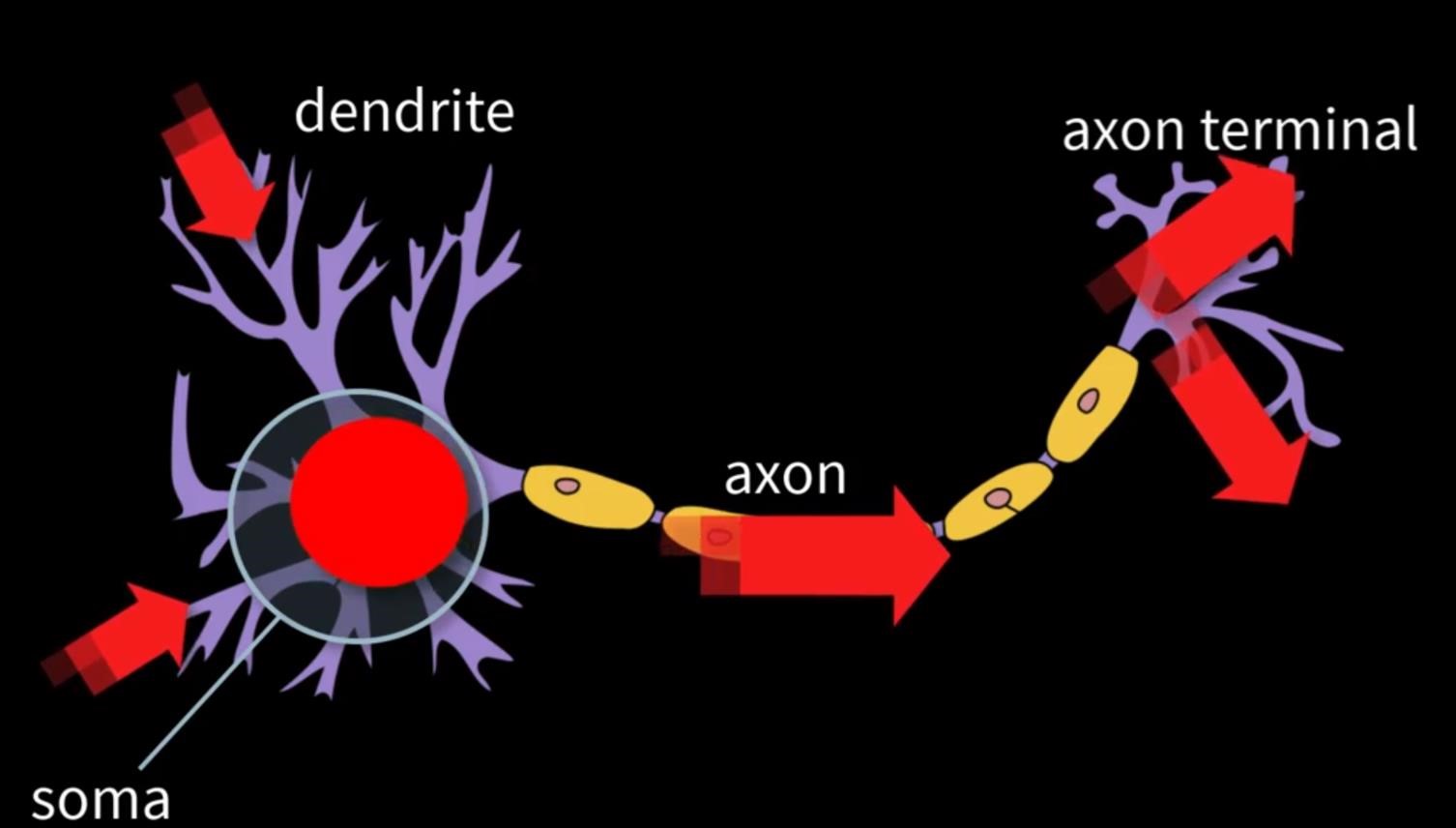


Module 3 is our final module of the game. This module may seem like a racing module to people who are playing our game for the first time. However, it`s racing with a twist. The twist is that there is no kind of racing. The added Artificial neural network-based car is basically just an illusion for the people in order to make them think that it`s actually a racing module. However, the requirement is displayed in red below on player`s screen stating that they should not hit wall at any cost. If they do so it`ll lead in a loss. Like in every module, winning this module also gives player 1 diamond. After reaching the finishing line, the player themself is taken to the next module and/or finishing screen.

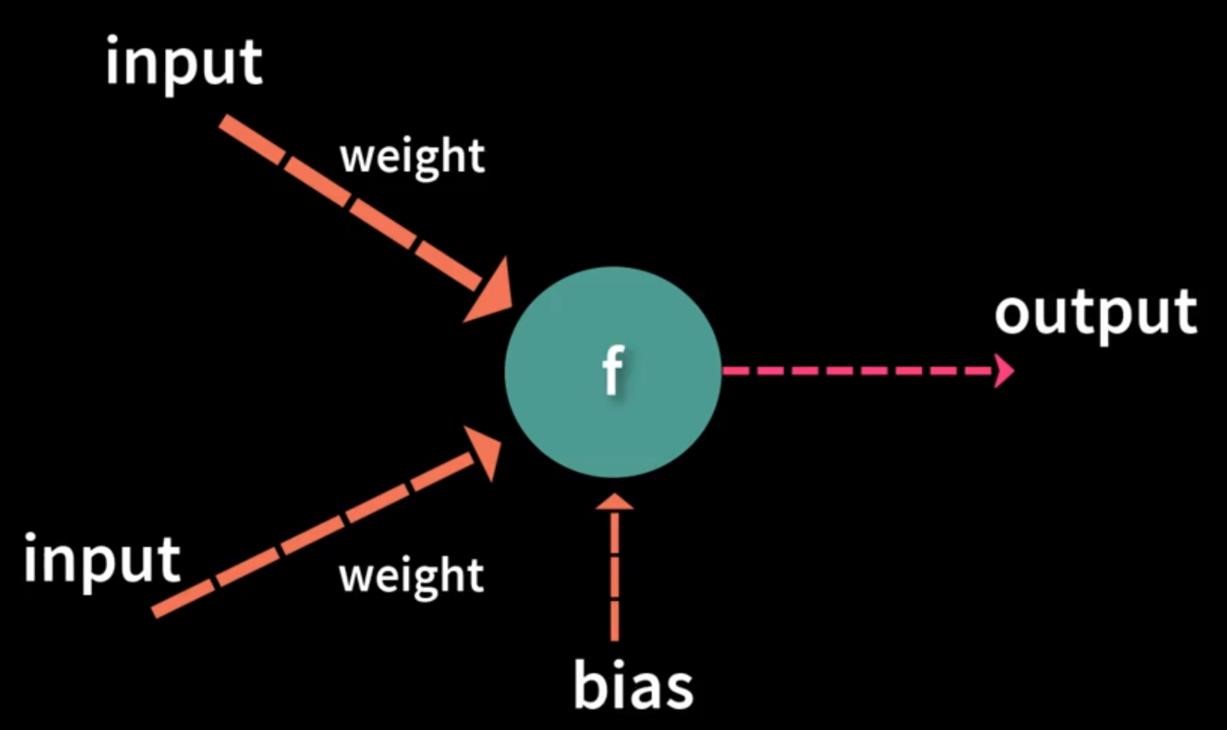


The Perceptron:

The perceptron is the fundamental algorithm behind the functioning of a neural network. It is a program that makes the building blocks of the human brain. The Neuron is a nerve cell not only found in the brain but elsewhere in the body throughout the nervous system. Neurons are connected together in a neural network and electrical impulses around the body.



The active learning in a neuron occurs through reinforcement, put simply, if you do something right and get rewarded then this is reinforcing the behavior and you are likely to do it again. If it`s a negative reward than you are less likely to do it again.

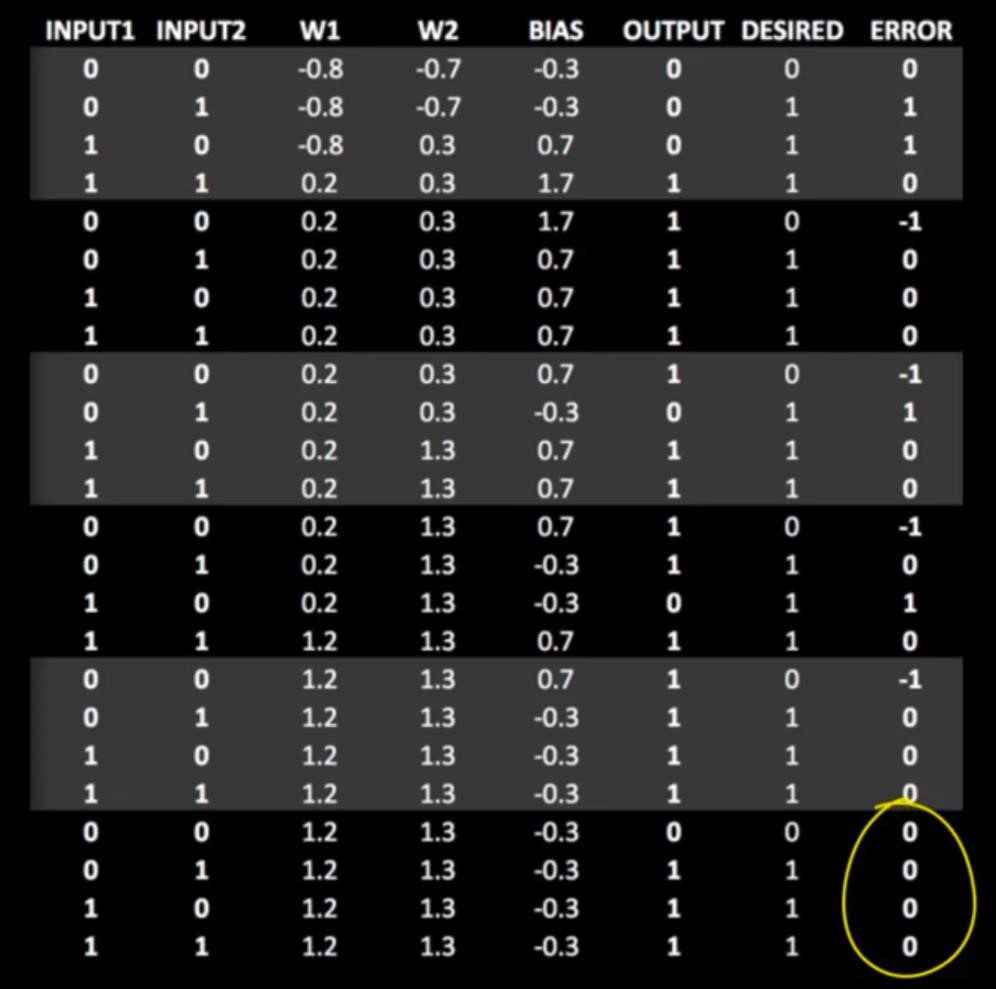


A perceptron is an algorithm that mimics the functioning of neuron. It takes input, processes that input with its own function and makes a decision as to whether it should fire or not. The weight values is where the learning takes place, these are multiplied by input and can be modified depending on whether or not the perceptron produces the correct output. The weight values hold what the perceptron has learned and is therefore the path that can be saved and reused for future use. They are essentially defined as how the perceptron performs.

Supposing a dataset of (OR FUNCTION):



Here, there are 2 errors and we know that the perceptron isn’t fully trained at this point. We then must run the training set through the perceptron again and we keep feeding the training set through the perceptron until it stops giving errors.



Each repetition of the training set through the perceptron is known as EPOCH. We can now be confident that if we ask the perceptron to calculate an OR it`ll produce the correct results. Our perceptron is now trained. The only values been modified in the algorithm being the weights. One can now save these weights and reload them into the perceptron at a later date when you would want it to calculate an OR.

The powerful idea behind machine learning is that you program once, but can retrain it many times.

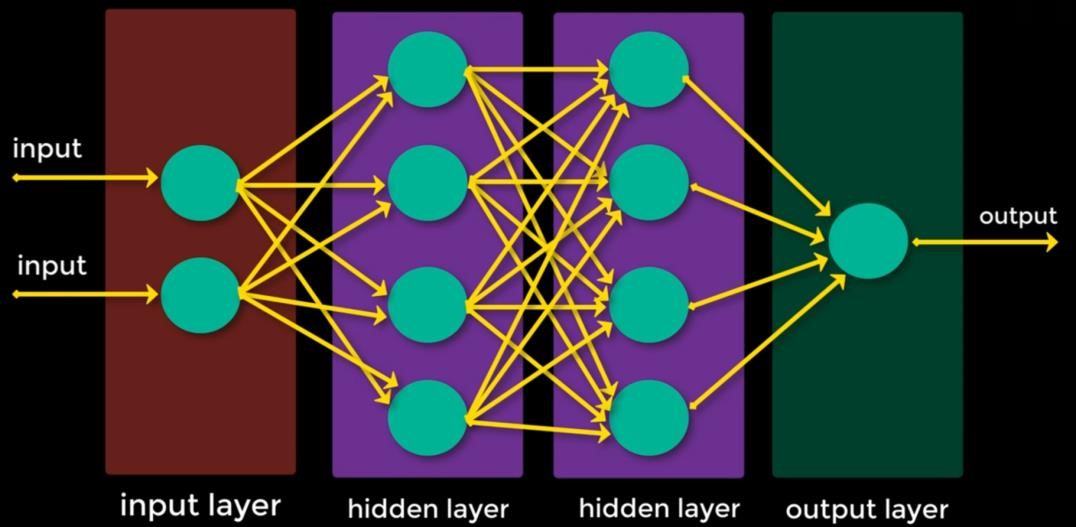
If we don’t want a perceptron to work as an OR calculator, we may train it for the AND. All we can do is change the training set; no reprogramming is required.

Artificial Neural Networks:

Single artificial neuron is mostly known as the perceptron. It`s a pretty cool concept, but to expand on its abilities we can link many of them together into something known as artificial neural networks. One may have heard of something known as ***deep learning***. ANN does nothing new but what perceptron does, all it has is a lot of perceptron, weights and outputs. It expands the capabilities of perceptron allowing the network to solve more complex problems.



In a basic neural network, the perceptron is organized into 3 or more layers. There is input layer, one or more hidden layers and then the output layer.



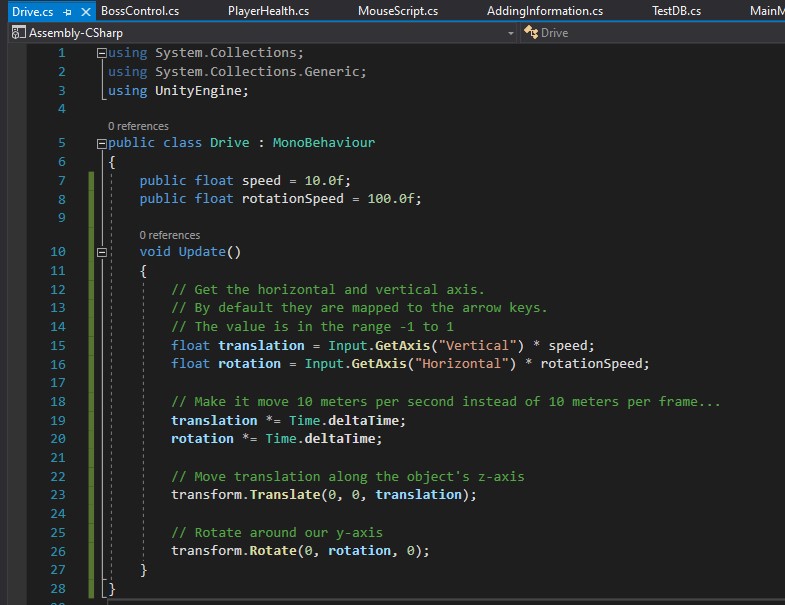
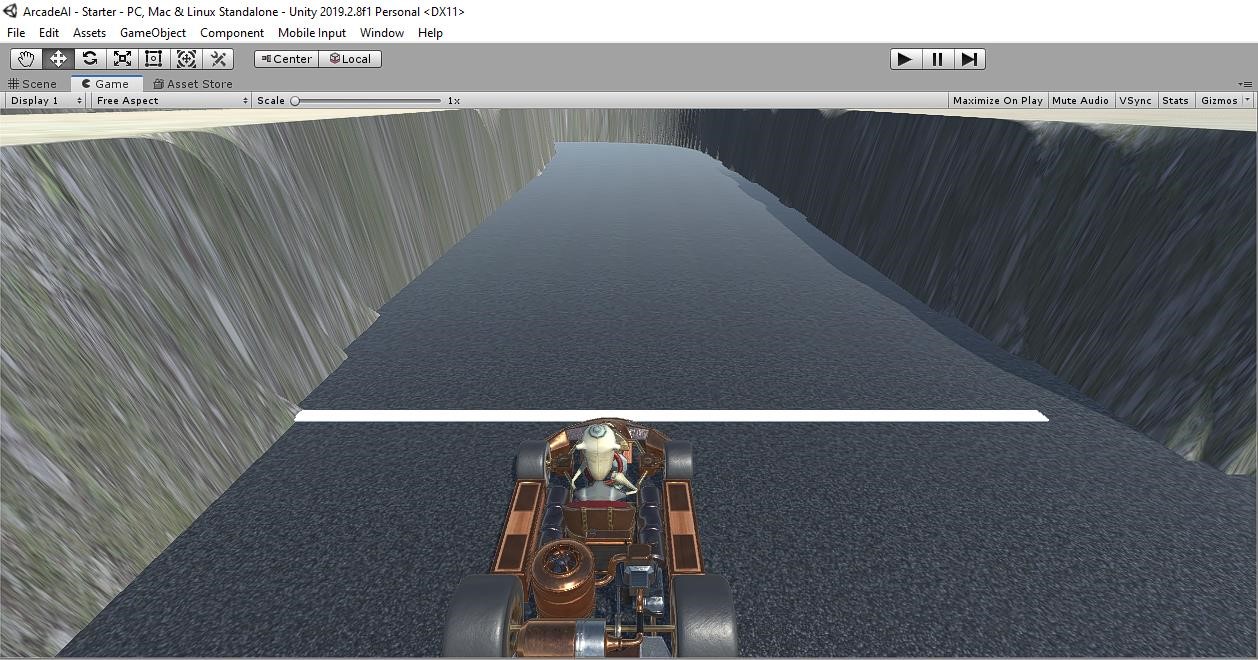
Entire network uses the same computational process as a single perceptron, each perceptron takes the input and multiplies it by it`s weight, processes it with its activation function and then passes the output to the perceptron’s in the next layer. This continues the output is obtained. The output is compared against the desired output from the training set. The error is calculated and then sent back up the layers in a process called ***back propagation*** to adjust all the weights in the neural network.



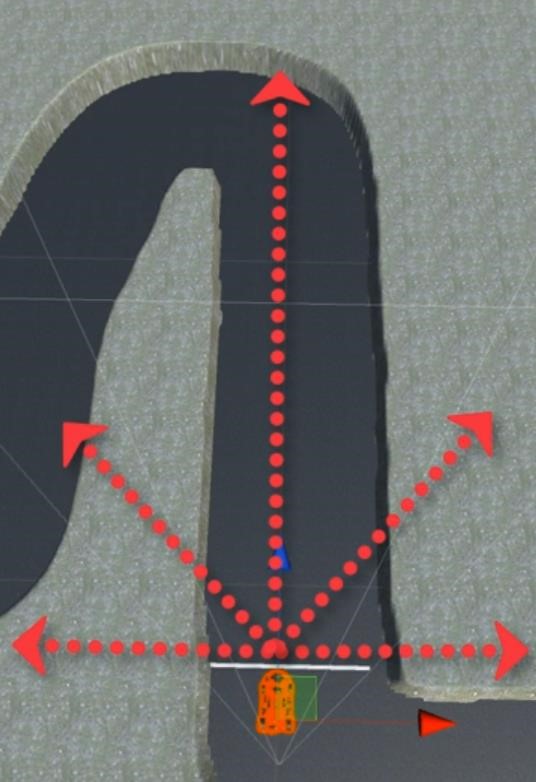
Why we used ANN (Deep learning)?

ANN are able to be applied to numerous problems including classification, data processing, robotics, computer control, and statistical analysis. They have been used in a vast number of systems from medical diagnosis to face identification. In games ANN isn’t very common because of their difficulty to train and inherit unpredictable behavior. However, they do appear from time to time.

Mechanics:

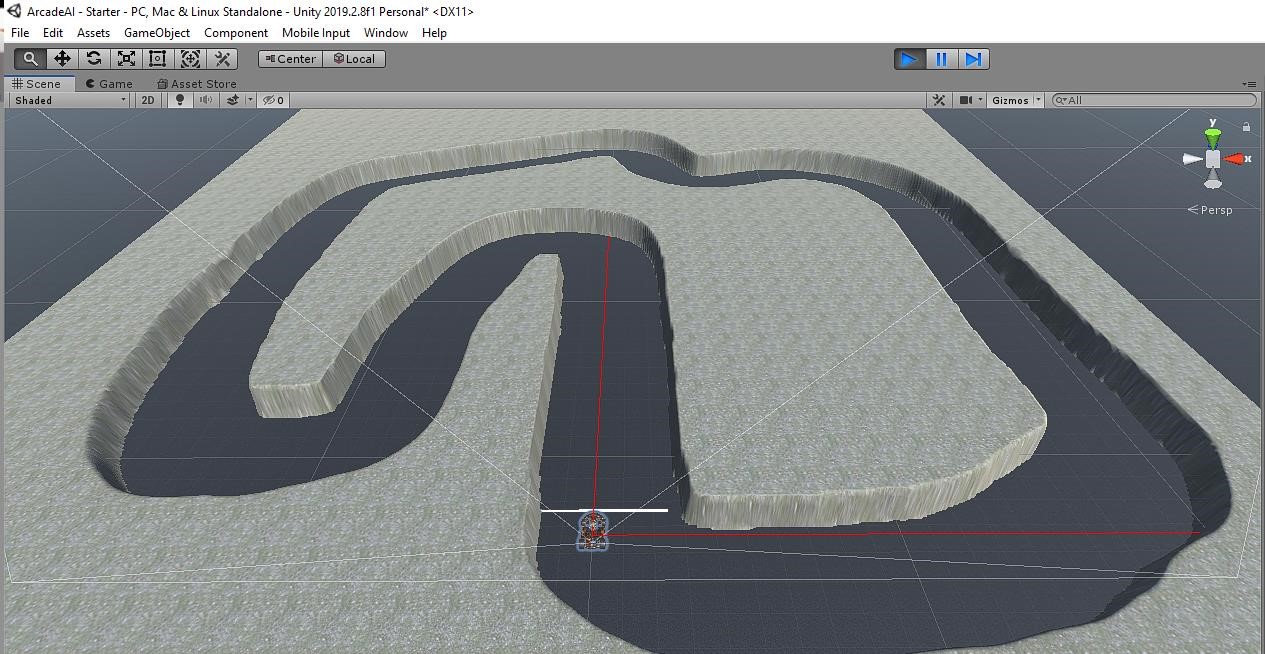


The 2 decisions that we as a player are making in this case is basically the control we have got in the game. The only control we have in this particular module of our game Arcade AI is to basically manipulate the vertical and horizontal axis in order to move the car (translation and the rotation). They are the only two inputs we are putting in as a player. As such, they are the 2 outputs that we want from our neural network, because we want our neural network to do the same thing that we are doing in order to control this kart (VEHICLE).



When we are driving, we are mostly concerned if someone is Infront of us, so it`s the distance. We can keep driving until we are about to hit something. However, when we are about to hit something, we are more interested in what`s to our left and right, and whether we can turn in that particular direction or not. In order to make our decision even strong, we also took under consideration the certain degrees ahead. In short, we would consider the forward, our left and right and what`s Infront of us at 45 degrees angle. We`ll do ray casting in order to know what`s at these sides of us.

Rays created:



As we can see the red rays going forward and in right, we could do it for all directions but this is just for demonstration purposes. Now, we know what`s 200 range ahead of us and to our right and at other places.

Code:

public float visibleDistance = 200.0f;

Debug.DrawRay(transform.position, this.transform.forward \* visibleDistance, Color.red); Debug.DrawRay(transform.position, this.transform.right \* visibleDistance, Color.red);

Performing ray cast:

RaycastHit hit;

float fDist = visibleDistance, rDist = visibleDistance,

lDist = visibleDistance, r45Dist = visibleDistance, l45Dist = visibleDistance;

//Foward if (Physics.Raycast(transform.position, this.transform.forward, out hit, visibleDistance))

{

fDist = hit.distance; //Returns where it hits.

}

//Right

if (Physics.Raycast(transform.position, this.transform.right, out hit, visibleDistance))

{

rDist = hit.distance;

}

//Left

if (Physics.Raycast(transform.position, -this.transform.right, out hit, visibleDistance))

{

lDist = hit.distance;

}

//Right 45

if (Physics.Raycast(transform.position, Quaternion.AngleAxis(45, Vector3.up) \* this.transform.right, out hit, visibleDistance))

{

r45Dist = hit.distance;

}

//Left 45

if (Physics.Raycast(transform.position, Quaternion.AngleAxis(45, Vector3.up) \* -this.transform.right, out hit, visibleDistance))

{

l45Dist = hit.distance;

}

These are all going to become inputs for our Artificial neural network, because these are the inputs of our original brain because we are converting what we see on the screen w.r.t our character. The outputs would be our rotation and translation.

So, what we are going to do is basically take all the input of the user (that is me) and store it inside a file in order for ANN to mimic that sort of behavior and provide our ANN our actual stored data as the data to train over.

Code;

string td = fDist + "," + rDist + "," + lDist + "," + r45Dist + "," + l45Dist + "," + translationInput + "," + rotationInput;

//Training Data

The inputs and outputs are basically stored in this string. Now, we are going to write it in a file. Going to add using System.IO; in order for using files and all.

In order to store all of the generated strings, we are going to create a List of string. In the end of our run we`ll write this list into a file.

List<string> collectedTrainingData = new List<string>(); collectedTrainingData.Add(td);

***Writing a method to write everything out in the file:***

StreamWriter tdf;

void Start()

{

string path = Application.dataPath + "/trainingData.txt"; tdf = File.CreateText(path);

}

This method at start is going to create a file with trainingData.txt (name). Writing to file using StreamWriter. Now, when we quit playing, we need to write everything to our file, I did this by introducing a new method:

void OnApplicationQuit()

{

foreach (string td in collectedTrainingData)

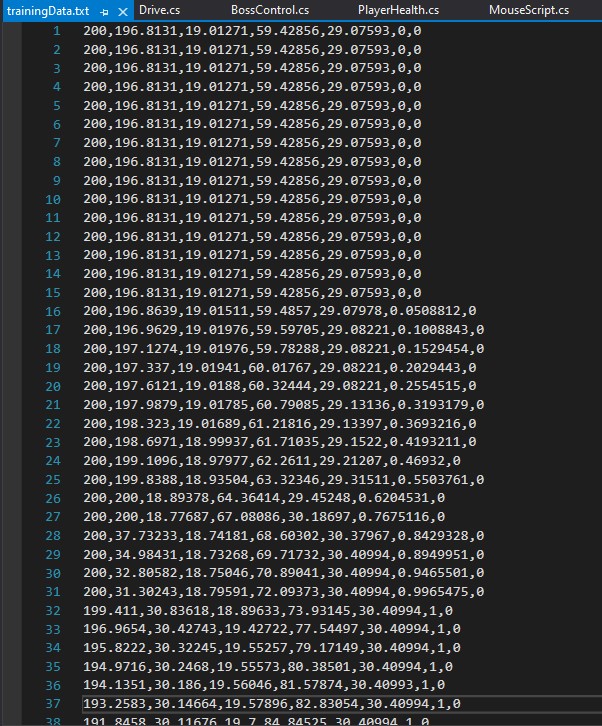
{

tdf.WriteLine(td);

}

tdf.Close(); }

Now I`m going to collect data by moving the vehicle and playing the module.



All of the driving information is stored inside this text file. But this isn’t as simple as it seems. Our neural network has a 5-dimensional space of inputs, it does illustrate difficulty for giving an optimal solution. We have got a lot of extra information (noise) that our neural network may not need in this particular file. We are going to use a process of normalization and also to round the numbers.

Performing Normalization (closer values are 1 and farthest values are 0):

Doing this so our data is more accurate than it was before and do not have all those abundant values and what not.

Also, we are going to round it to its nearest 0.5.

float Round(float x)

{

return (float)System.Math.Round(x, System.MidpointRounding.AwayFromZero) / 2.0f;

}

Normalization:

float fDist = 0, rDist = 0,

lDist = 0, r45Dist = 0, l45Dist = 0;

//Foward

if (Physics.Raycast(transform.position, this.transform.forward, out hit, visibleDistance))

{

fDist = 1 - Round(hit.distance/visibleDistance); //Performing normalization, will give something between 0 and 1.

}

//Right

if (Physics.Raycast(transform.position, this.transform.right, out hit, visibleDistance))

{

rDist = 1 - Round(hit.distance / visibleDistance);

}

//Left

if (Physics.Raycast(transform.position, -this.transform.right, out hit, visibleDistance))

{

lDist = 1 - Round(hit.distance / visibleDistance);

}

//Right 45

if (Physics.Raycast(transform.position, Quaternion.AngleAxis(45, Vector3.up) \* this.transform.right, out hit, visibleDistance))

{

r45Dist = 1 - Round(hit.distance / visibleDistance);

}

//Left 45 if (Physics.Raycast(transform.position, Quaternion.AngleAxis(45, Vector3.up) \* -this.transform.right, out hit, visibleDistance))

{

l45Dist = 1 - Round(hit.distance / visibleDistance);

}

The last thing we are going to do is basically to prevent our list from adding useless values, we are going to do so by testing whether or not that particular string already exists inside our list.

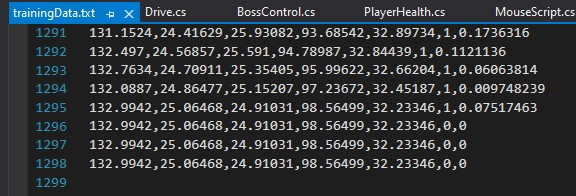
if (!collectedTrainingData.Contains(td))

{

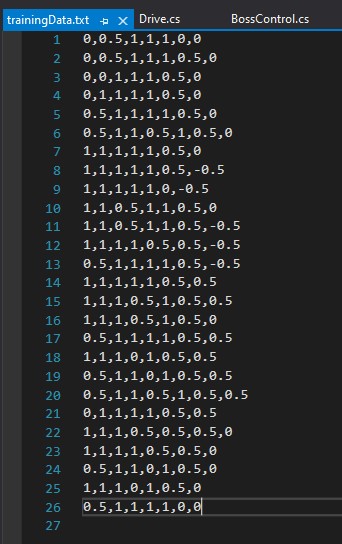
collectedTrainingData.Add(td);

}

Last time the training data was 1300 lines (approximately),



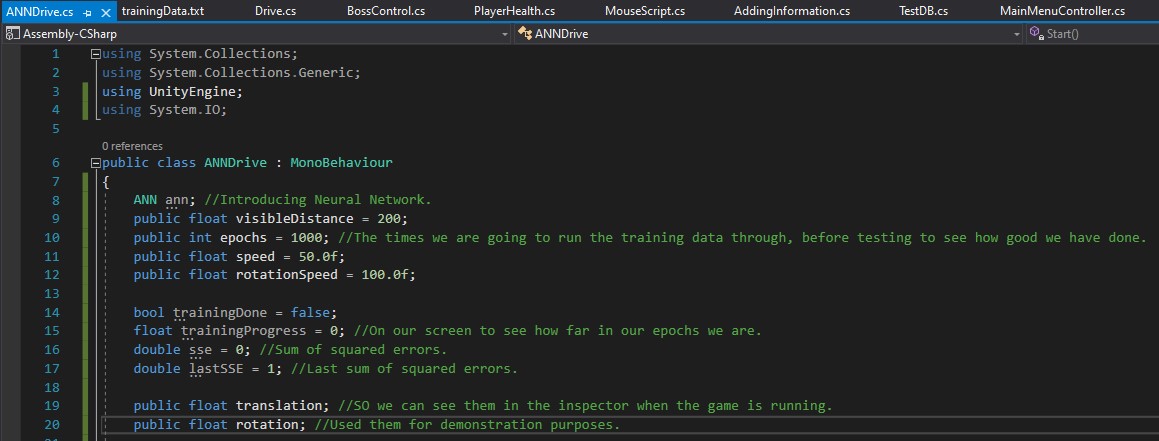
We reduced this by 10000x basically (hehe):



Only 26 lines. This is going to make our neural network train faster and more accurate even in some cases.

Training with player data:

Making a new script and introducing variables along with the neural network we created;



We can drive it on any track using this training, not specifically on the track of our game (Arcade AI). Until training Done Boolean is set to true, we wont be able to move to update.

We are going to use 10 number of neurons as it`s double the number of inputs that is 5. The rule of thumb is that says double the number of inputs and find a number between them which ends up being 10 which is the end of them. Did a lot of trial and error in order to find this value of 10 that works the best, the more neurons would just work the same way.

For demonstration, going to use:

void OnGUI()

{

GUI.Label(new Rect(25, 25, 250, 30), "SSE: " + lastSSE);

GUI.Label(new Rect(25, 40, 250, 30), "Alpha: " + ann.alpha);

GUI.Label(new Rect(25, 55, 250, 30), "Trained: " + trainingProgress);

}

Introducing a method where all the work would be happening named as (LoadTrainingSet) that would be a coroutine (IEnumerator):

First, we are going to include the path of our file. If the file exists, we are going to count all of the lines. Further demonstration is done using comments:

ANN ann; //Introducing Neural Network.

public float visibleDistance = 200; public int epochs = 1000; //The times we are going to run the training data through, before testing to see how good we have done.

public float speed = 50.0f; public float rotationSpeed = 100.0f;

bool trainingDone = false; float trainingProgress = 0; //On our screen to see how far in our epochs we are.

double sse = 0; //Sum of squared errors. double lastSSE = 1; //Last sum of squared errors.

public float translation; //SO we can see them in the inspector when the game is running. public float rotation; //Used them for demonstration purposes.

// Start is called before the first frame update void Start()

{

ann = new ANN(5, 2, 1, 10, 0.5); //5 inputs, 2 outputs, hidden layers = 1, number of neurons in layer = 10, alpha value = 0.5.

StartCoroutine(LoadTrainingSet());

}

void OnGUI()

{

GUI.Label(new Rect(25, 25, 250, 30), "SSE: " + lastSSE);

GUI.Label(new Rect(25, 40, 250, 30), "Alpha: " + ann.alpha);

GUI.Label(new Rect(25, 55, 250, 30), "Trained: " + trainingProgress);

}

IEnumerator LoadTrainingSet()

{

string path = Application.dataPath + "/trainingData.txt"; string line;

if (File.Exists(path))

{

int lineCount = File.ReadAllLines(path).Length;

StreamReader tdf = File.OpenText(path); //Going to open our training data.

List<double> calcOutputs = new List<double>(); //Gets what the neural network is calculating to send back.

List<double> inputs = new List<double>(); //Input values.

List<double> outputs = new List<double>(); //Output values.

for (int i = 0; i < epochs; i++) //Going to go through this code 1000 of times.

{

//Setting file pointer to the beggining of the file.

sse = 0; tdf.BaseStream.Position = 0; //This is so when we loop to the end, we can come back to the start.

while ((line = tdf.ReadLine()) != null) //Reading 1 line at a time until it runs out of lines.

{

string[] data = line.Split(','); //Taking line from the file and going to split it on the comma.

//If nothing to be learned, we going to ignore this line. float thisError = 0; //Calculating error that we get from our training.

if (System.Convert.ToDouble(data[5]) != 0 && System.Convert.ToDouble(data[6]) != 0) //Data in position 5 and 6 == 0 that is translation and rotation.

{

inputs.Clear(); outputs.Clear(); inputs.Add(System.Convert.ToDouble(data[0])); //Adding values for our distance. Straight. 5 input values coming out from our training data. inputs.Add(System.Convert.ToDouble(data[1])); //Right inputs.Add(System.Convert.ToDouble(data[2])); //Left inputs.Add(System.Convert.ToDouble(data[3])); //Right45 inputs.Add(System.Convert.ToDouble(data[4])); //Left45

double o1 = Map(0, 1, -1, 1, System.Convert.ToSingle(data[5])); //Output values that is translation. outputs.Add(o1); double o2 = Map(0, 1, -1, 1, System.Convert.ToSingle(data[6])); //Output values that is rotation.

outputs.Add(o2);

calcOutputs = ann.Train(inputs, outputs); //training with inputs and outputs. We are getting 2 outputs (Translation and rotation). thisError = ((Mathf.Pow((float)(outputs[0] - calcOutputs[0]), 2) + Mathf.Pow((float)(outputs[1] - calcOutputs[1]), 2))) / 2.0f;

//Calculating sum of squared errors on the 1st output and as well as on the 2nd output.

}

sse += thisError;

}

trainingProgress = (float)i / (float)epochs; //i is epochs loop counter. Going to give percentage value trained. sse /= lineCount; //Giving us average of errors.

lastSSE = sse;

yield return null;

}

}

trainingDone = true; }

Making the update function, that is super difficult.

Now all of this is calculated using the brain of neural network, rather than the human brain. It must have the same senses that is basically the distance from all sides. Comments added for more demonstration.

void Update()

{

if (!trainingDone) //This is just preventing update from starting and trying to drive the car when the neural network hasnt finished training.

{

return;

}

List<double> calcOutputs = new List<double>();

List<double> inputs = new List<double>();

List<double> outputs = new List<double>();

RaycastHit hit; float fDist = 0, rDist = 0,

lDist = 0, r45Dist = 0, l45Dist = 0;

//Foward

if (Physics.Raycast(transform.position, this.transform.forward, out hit, visibleDistance))

{

fDist = 1 - Round(hit.distance / visibleDistance); //Performing normalization, will give something between 0 and 1.

}

//Right

if (Physics.Raycast(transform.position, this.transform.right, out hit, visibleDistance))

{

rDist = 1 - Round(hit.distance / visibleDistance);

}

//Left

if (Physics.Raycast(transform.position, -this.transform.right, out hit, visibleDistance))

{

lDist = 1 - Round(hit.distance / visibleDistance);

}

//Right 45

if (Physics.Raycast(transform.position, Quaternion.AngleAxis(45, Vector3.up) \* this.transform.right, out hit, visibleDistance))

{

r45Dist = 1 - Round(hit.distance / visibleDistance);

}

//Left 45

if (Physics.Raycast(transform.position, Quaternion.AngleAxis(45, Vector3.up) \* -this.transform.right, out hit, visibleDistance))

{

l45Dist = 1 - Round(hit.distance / visibleDistance);

}

inputs.Add(fDist); //Sending these values as input in our neural network. inputs.Add(rDist); inputs.Add(lDist); inputs.Add(r45Dist); inputs.Add(l45Dist); outputs.Add(0); //Outputs are just placeholder, not really gonna use them. outputs.Add(0);

calcOutputs = ann.CalcOutput(inputs, outputs); //calculating outputs, will be in range of 0 and 1 because that is what I wanted.

float translationInput = Map(-1, 1, 0, 1, (float)calcOutputs[0]); //Mapping them and converting them back to between -1 and 1. float rotationInput = Map(-1, 1, 0, 1, (float)calcOutputs[1]); //This is same as how bhuman player plays and presses keys.

//Now we are getting it from neural network for our NPC kart (car).

translation = translationInput \* speed \* Time.deltaTime; //Calculated exactly the same way as they were before. rotation = rotationInput \* rotationSpeed \* Time.deltaTime;

this.transform.Translate(0, 0, translation); this.transform.Rotate(0, rotation, 0);

}

Got a problem where my neural network stuck in the local optima and not was able to get out.

Using ***adaptive learning*** in order to fix this problem.

string currentWeigths = ann.PrintWeights(); //Going to store current weights from current state of neural networks.

//if sse isnt better then reload previous set of weights and declare alpha.

if (lastSSE < sse)

{

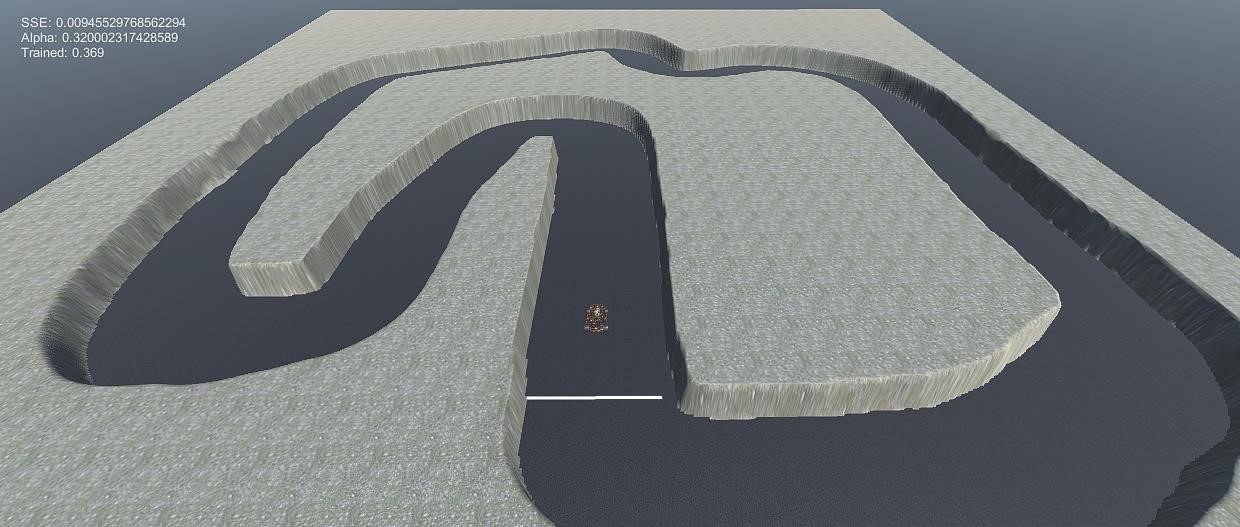
ann.LoadWeights(currentWeigths); ann.alpha = Mathf.Clamp((float)ann.alpha - 0.001f, 0.01f, 0.9f); //Going to decrease it by 0.001, clamp to make sure it doesnt go negative or more than 1.

}

else //Increase alpha

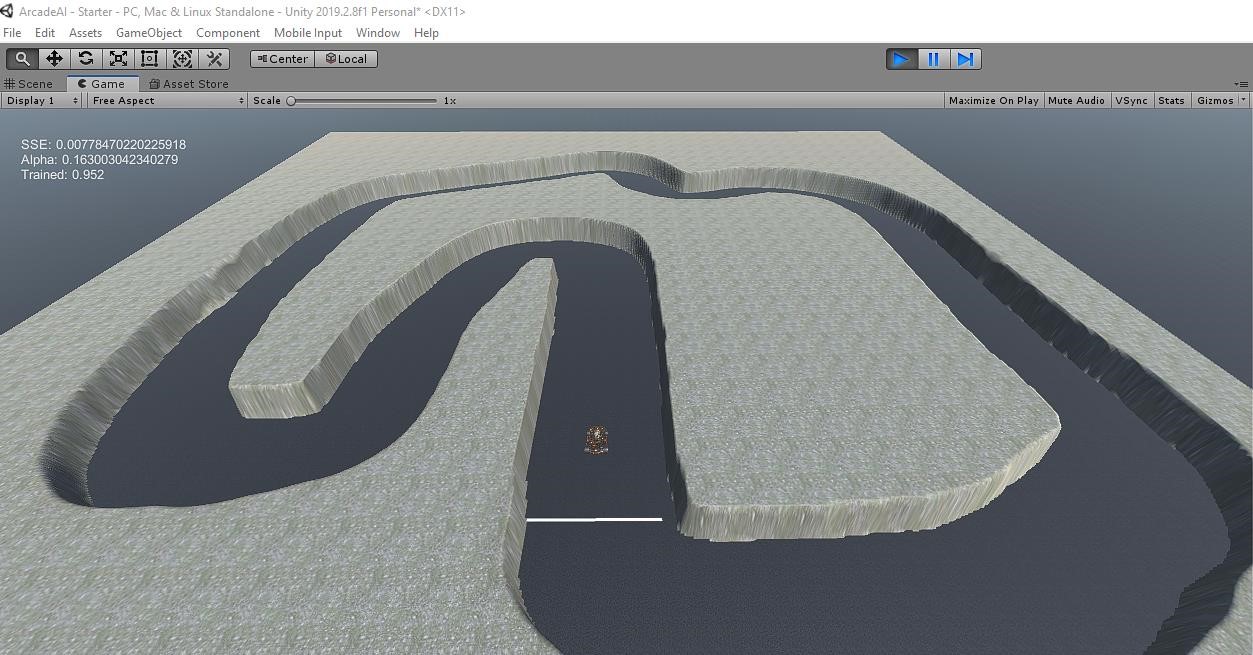
{

ann.alpha = Mathf.Clamp((float)ann.alpha + 0.001f, 0.01f, 0.9f); lastSSE = sse; }



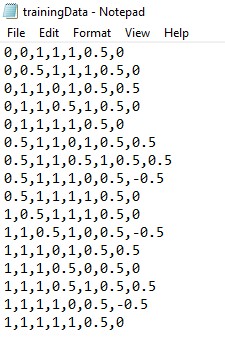
Performing tests in order to find the lowest SSE so that it`s train almost close to perfect. It is pretty difficult to find low SSE.

Adding code that will save the weights that we have come up with the ANN, because I don’t want to keep training the neural network all the time in order to find the required solution.

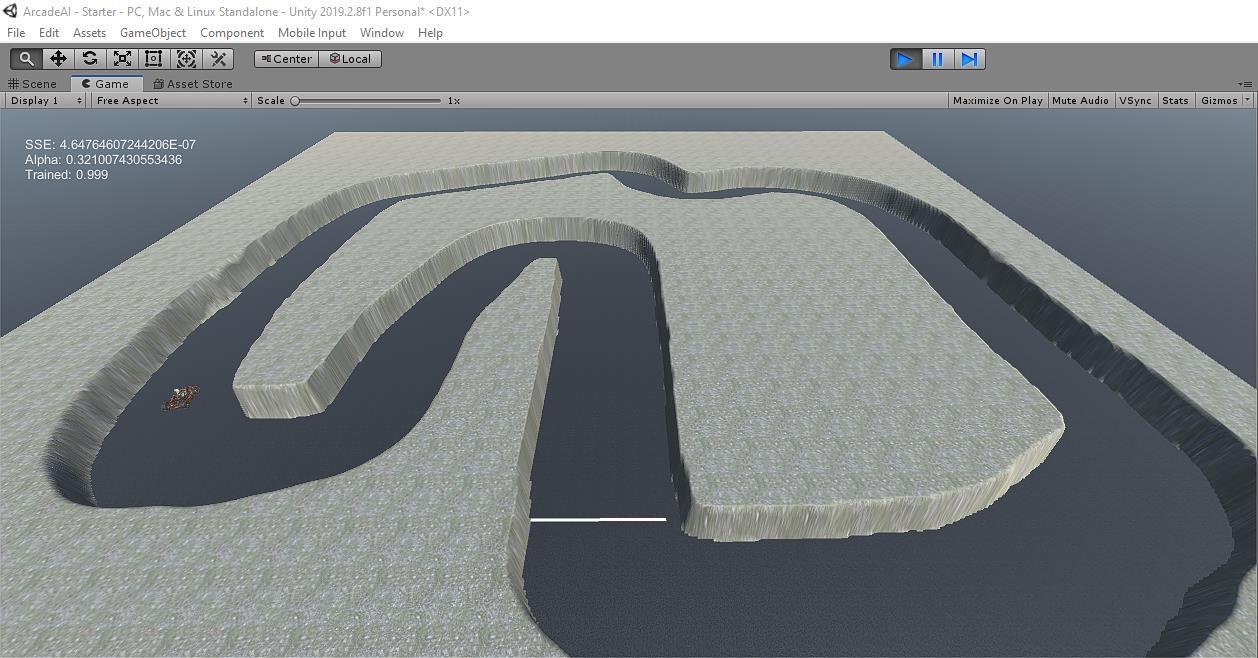


Finding an almost very good solution, saving it in weights.txt for future use.

The best training data that I got after intense amount of training:



Finally, after extensive (intense training), we were able to train our NPC using artificial neural network with an SSE value of exponent -7 that is extremely low.



**Description of the user interface:**

**Start Screen:**



This will consider 2 things. The first one would be the existing users, for that it`ll just use their previous information in order to maintain their profile. However, for new users it will map name 0 0 0 values in database because games played and diamonds attained = 0, at first.

**Loading Screen:**



This screen will load in order to transition from 1 scene to another. It uses co-routines and is displayed for 3 seconds.

**Main Menu Screen:**



**Help:**

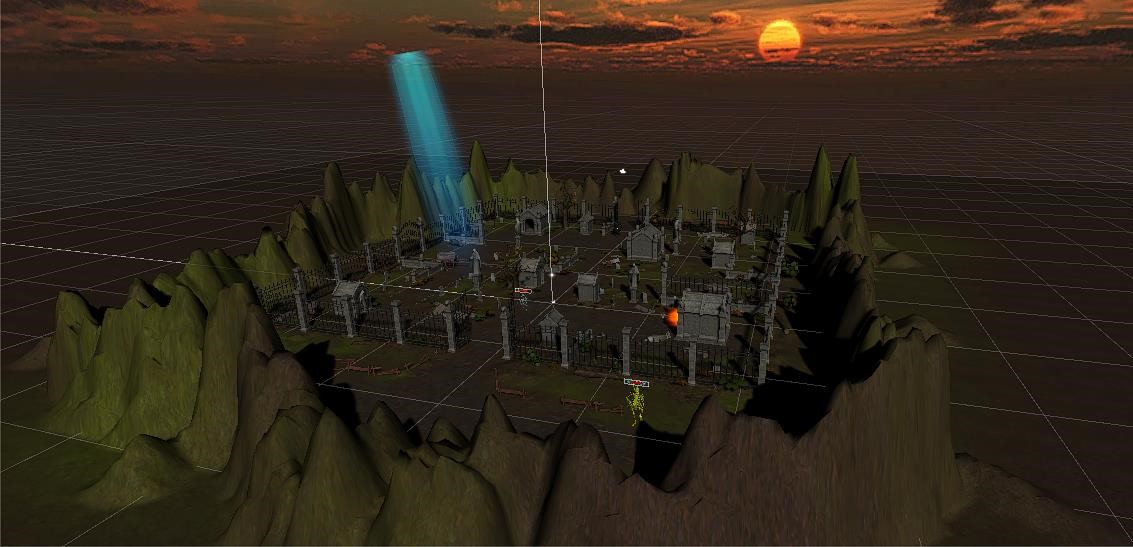


**Information:**



**1st Module:**





## Module 2:



Need to find the person (girl) displayed on top-right.



Diamonds attained, as we can see on the top-left.

## Module 3:





**Finish Screen:**



An assessment of the player, diamonds attained = 3, number of games played = 1, name = Moeez and time taken in order to complete the game or attain those 3 diamonds = 61 seconds.

Before:



After:



**Safety warning:**

L-1: Initially the game can be played by the citizens of Pakistan and some other Asian countries, after that it is expected that the game will be promoted to the citizens of American and European countries.

L-2: The game can only be played in the single-player mode.

L-3: For accelerated 3D graphics (1.5GB of NVIDIA graphics card) is required as there is a lot to render in our 3D game.

L-4: Keyboard, Mouse must be attached in order for person to play professionally.

L-5: A RAM of 4GB minimum is required.

L-6: Choose a time slot to practice every day, do not get ADDICTED!

**Installation environment:**

* Download the Full Game Installer that includes the entire game data.
* Once the download has completed, start the Installation and follow the on screen instructions to complete the installation procedure.
* After completing installation, double click on A3.exe to start the Game.
* Please note that you will have to update to the Latest client version before proceeding.
* I`m using unity version **2019.2.8f1** (any other version may provide some issues).

**GitHub Link:**<https://github.com/moeez-ahmed/ArcadeAI>