NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTION)

(AFFILIATED TO VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM, APPROVED BY AICTE & GOVT.OF KARNATAKA)

MINOR PROJECT REORT

ON

SELF DRIVING CAR USING ML-AGENTS

VISHNU RAJENDRAN	1NT15CS208
SIDDHARTH V	1NT15CS177
ZUBAIDA TABASSUM	1NT15CS212
T KISHORE	1NT15CS081

Programming Assignment Report Submitted as partial fulfillment of the requirement for the SELF DRIVING CAR USING ML-AGENTS course in VI Semester of BE Degree in Computer Science and Engineering

Department of Computer Science and Engineering

NitteMeenakshi Institute of Technology,

Yelahanka, Bangalore– 560064

October-November 2016

NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTION)

(AFFILIATED TO VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM, APPROVED BY AICTE & GOVT.OF KARNATAKA)

CERTIFICATE

This is to certify that the Minor Project on

SELF DRIVING CAR USING ML-AGENTS

is an authentic work carried out by

VISHNU RAJENDRAN	1NT15CS208
SIDDHARTH V	1NT15CS177
ZUBAIDA TABASSUM	1NT15CS212
T KISHORE	1NT15CS081

In partial fulfillment of the requirements for the award of BE Degree in Computer Science & Engineering during the year 2018-2019.

Guide	Head of the Department
Dr. Saroja Devi	Dr. Thippeswamy M N
Signature of examiner	Signature of examiner

ACKNOWLEDGEMENT

The credit for the successful completion of this programming assignment work goes beyond our own work, to those people who have always been with us throughout. And we take this opportunity to express our heartfelt gratitude to each one of them. We express our solicit thanks to our HOD, **Dr. THIPPESWAMY M N**, Department of Computer Science and Engineering for giving us an opportunity to carry out the project. We express our deepest gratitude to our internal guide, **Dr. SAROJA DEVI**, Department of Computer Science and Engineering, for her valuable suggestions, support, encouragement and constructive criticism that enabled us to complete this project successfully. We extend our thanks to faculty of Department of CSE for their effort and endurance to bring out the best in all of us. Finally, we would like to thank our beloved parents, friends and dear ones for standing with us in all times.

ABSTRACT

Machine Learning is a study in Computer Science in which a Computer is programmed to perform a specific task, but not by explicitly programming it to. That is the computer learns what to do with the given data. But before it can do so, we need to train the computer using training sets or environments giving it appropriate observations, slowly by observing the data it receives it tries to recognize patterns in data and react accordingly.

Machine Learning is a relatively new area in computer science, as the power required for such operations are really high, and only in last decade did our computers become able handle such loads.

In our project we use machine learning to try to train a car (virtual) to drive on its own without any input from the user. This was achieved by using Unity 3D Game Engine, a software which is used to build simulations, short movies and games and their latest plugin Unity ML-Agents(beta 0.3). The Project was trained using Google's TensorFlow.

Table of Contents

1. INTRODUCTION

Driverless cars are now being developed by almost every automotive company (Tesla Motors) and some non-automotive companies (Google).But these cars need to be trained for it have an use in the future and hence is done so. But releasing an automobile into the streets among civilians is just a call for trouble, due to the humongous amount of variables that are present during an normal driving session is tedious to program and tough to anticipate. This is where machine learning come into play, By training the car using machine learning, gives us the opportunity to give the car the power to improve itself, thereby eliminating most of the variables at play. But yet again releasing such a machine into the public to train would be dangerous as road condition, natural disasters , and mechanical failure can lead to someone getting hurt or damage the setup.

There are two ways to solve this problem, one have a city where some variables such as human activity is removed or two find a place where such problem can be avoided altogether. Now Google has taken the first solution and has built a city, all with traffic systems, weather system and there is continuously being trained there. Unfortunately for some companies such as Tesla Motors, such setups would be a poor investment. And so we think the solution to this problem is virtual. By training the car in an virtual environment, we are able to avoid safety breaches, have full control of the environment its been trained on and be able to monitor the training at a very detailed level. And so we tried to implement the same using Unity 3D Game Engine and their new ML-Agents plugin currently in beta 0.3, and the results are satisfying.

2.LITERATURE SURVEY

An autonomous car (also known as a driverless car and a self-driving car) is a vehicle that is capable of sensing its environment and navigating without human input.

Autonomous cars combine a variety of techniques to perceive their surroundings, including radar, laser light, GPS, odometry, and computer vision. Advanced control systems interpret sensory information to identify appropriate navigation paths, as well as obstacles and relevant signage.

The potential benefits of autonomous cars include reduced mobility and infrastructure costs, increased safety, increased mobility, increased customer satisfaction, and reduced crime. These benefits also include a potentially significant reduction in traffic collisions, resulting injuries, and related costs, including less need for insurance. Autonomous cars are predicted to increase traffic flow, provide enhanced mobility for children, the elderly, disabled, and the poor, relieve travelers from driving and navigation chores, lower fuel consumption, significantly reduce needs for parking space, reduce crime, and facilitate business models for transportation as a service, especially via the sharing economy. This shows the vast disruptive potential of the emerging technology.

In spite of the various potential benefits to increased vehicle automation, there are unresolved problems, such as safety, technology issues, disputes concerning liability, resistance by individuals to forfeiting control of their cars, customer concern about the safety of driverless cars, implementation of a legal framework and establishment of government regulations; risk of loss of privacy and security concerns, such as hackers or terrorism; concern about the resulting loss of driving-related jobs in the road transport industry; and risk of increased suburbanization as travel becomes less costly and time-consuming. Many of these issues arise because autonomous objects, for the first time, would allow computer-controlled ground vehicles to roam freely, with many related safety and security concerns.

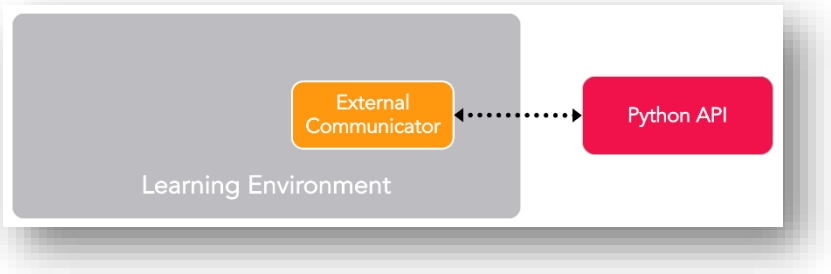
Experiments have been conducted on automating driving since at least the 1920s; promising trials took place in the 1950s. Tsukuba Mechanical Engineering Lab in Japan created the first autonomous, intelligent vehicle in 1977. It tracked white street markers and achieved speeds up to 30 kilometres per hour (19 mph) (tech- faq.com). Autonomous prototype cars appeared in the 1980s, with Carnegie Mellon University's Naval and ALV projects funded by DARPA starting in 1984 and Mercedes-Benz and Bundeswehr University Munich's EUREKA Prometheus Project in 1987. By 1985 the ALV had demonstrated self-driving spe eds on two-lane roads of 31 kilometers per hour (19 mph) with obstacle avoidance added in 1986 and off-road driving in day and nighttime conditions by 1987. From the 1960s up through the second DARPA Grand Challenge in 2005, autonomous vehicle research in the U.S. was primarily funded by DARPA, the US Army and the U.S. Navy yielding incremental advances in speeds, driving competence in more complex conditions, controls, and sensor systems. Since then, numerous companies and research organizations have developed prototypes.

In 2017, Audi stated that its latest A8 would be autonomous at speeds of up to 60 kilometers per hour (37 mph) using its "Audi AI". The driver would not have to do safety checks such as frequently gripping the steering wheel. The Audi A8 was claimed to be the first production car to reach level 3 autonomous driving, and Audi would be the first manufacturer to use laser scanners in addition to cameras and ultrasonic sensors for their system. In November 2017, Waymo announced that it had begun testing driverless cars without a safety driver in the driver position; however there is still a n employee in the car. In February 2018, Waymo announced that its test vehicles had traveled autonomously for over 5,000,000 miles (8,000,000 km).

3.DESIGN

ML-Agents

The ML-Agents toolkit is a Unity plugin that contains three high-level components Learning Environment - which contains the Unity scene and all the game characters. Python API - which contains all the machine learning algorithms that are used for training. External Communicator - which connects the Learning Environment with the Python API. It lives within the Learning Environment.



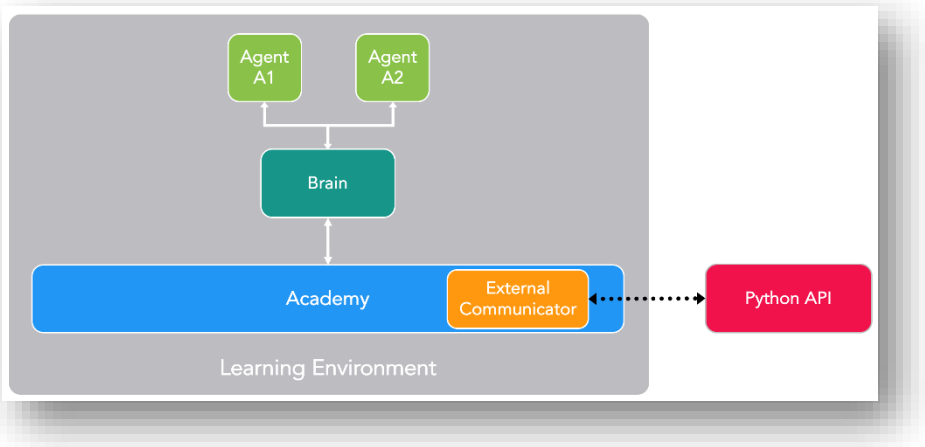
Learning environment

The Learning Environment contains three additional components that help organize the Unity scene:

- **Agents** - which is attached to a Unity Game Object and handles generating its observations, performing the actions it receives and assigning a reward when appropriate.

- Brains - which encapsulates the logic for making decisions for the Agent.

- Academy - which orchestrates the observation and decision making process.



Proximal Policy Optimization (PPO)

ML-Agents uses a reinforcement learning technique called Proximal Policy Optimization (PPO). PPO uses a neural network to approximate the ideal function that maps an agent's observations to the best action an agent can take in a given state. The ML-Agents PPO algorithm is implemented in TensorFlow and runs in a separate Python process (communicating with the running Unity application over a socket).

proximal policy optimization (PPO), have some of the benefits of trust region policy optimization (TRPO), but they are much simpler to implement, more general, and have better sample complexity (empirically).

The hyper-parameters PPO uses for ML-Agents is given below. These values are stored under "trainer_config.yaml" file under the python directory. Every Brain object can have a set of hyper-parameters, or it can the default values assigned by unity.

```yaml
carBrain:
    trainer: ppo
    batch_size: 1024
    beta: 5.0e-3
    buffer_size: 1024
    epsilon: 0.2
    gamma: 0.99
    hidden_units: 128
    lambd: 0.95
    learning_rate: 2.0e-4
    max_steps: 5.0e6
    memory_size: 1024
    normalize: true
    num_epoch: 3
    num_layers: 2
    time_horizon: 64
    sequence_length: 64
    summary_freq: 1000
    use_recurrent: false
    use_curiosity: false
    curiosity_strength: 0.01
    curiosity_enc_size: 128
```

Here trainer, sets the value of the algorithm used to train the agents. Usually either its ppo for reinforcement training or imitation for imitation training. **max_steps** corresponds to how many steps of the simulation (multiplied by frame-skip) are run during the training process**. num_layers** define the no .of hidden layers, usually between a range of 1 – 3, where 1 is sufficient for simple scenarios and 3 for complex ones. For us 2 hidden layers is sufficient. **buffer_size** corresponds to how many experiences (agent observations, actions and rewards obtained) should be collected before we do any learning or updating of the model.

4.IMPLEMENTATION

The Project was implemented using Unity and ML-Agents. The first part of our implementation is building our essentials such as a vehicle and the terrain. these are explained in detail in the following sections.

4.1Cars and Terrains

The design of the vehicle was simple enough, all we needed in our first trials was a body with 4 wheels and some sensors to retrieve distance data. This was achieved with a cuboid with rigid body property serving as the body, four wheel colliders (colliders are invisible boxes used to detect collisions or contact or apply constraints) and 5 cubes representing sensors at front center, left, right and middle left and right. The observations for the brain from this vehicle are the distance data an the cars current position and its relative position with respect to the target(Destination point).

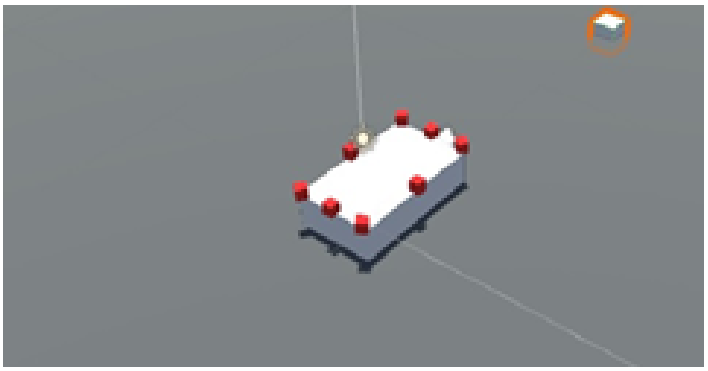
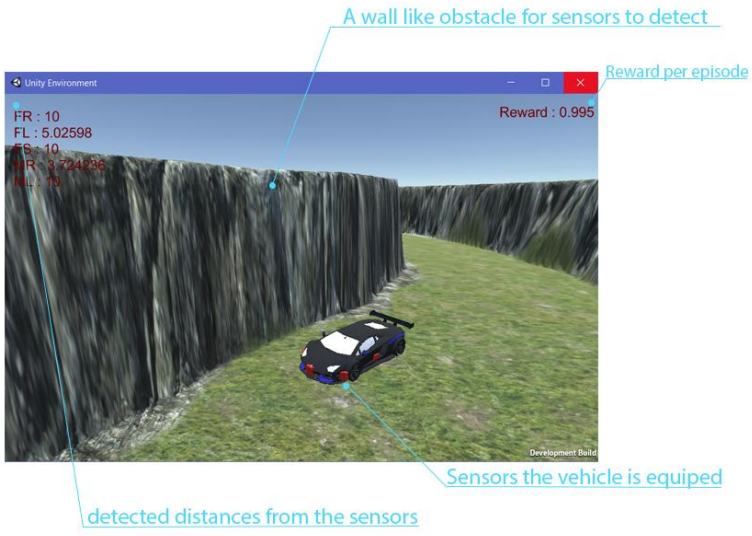


Fig. An early version of the car with eight sensors



Later stages the car model is replaced ,some more colliders are added and the sensor size was decreased from 8 to 5.

After close to 30 lakh iterations depending on the track ,the vehicle is now able to successfully traverse a curved route using sensors that detect distance between the vehicle and area around the vehicle with a max range of 20 unity space units. The agent script uses these sensor data as observations that are fed to the external neural network during training along with the vehicle's position in vector3 (x, y and z) format and the

velocity of the vehicle in the z-direction (forward direction of the vehicle).In the following pages the sensors and the agent scripts will be explained in detail.

The figure shows an episode of the training process, as shown the prototype model has been replaced with a model of a sports car, with red cubes protruding from the body, these cubes serves us the purpose of sensors and detect anything in front of it. The left side of the screen displays distances measured by these sensors with a max range of 10, that is any object that is less than 10 units from the vehicle will be detected. The right side displays the reward the brain receives per episode and it is calculated according to some conditions. This version of training took a full 10 Lakh iterations to learn completely.

Tracks

Tracks are generated using heightmaps and Unity's Terrain, the heightmap is a black and white RAW image where pure white is maximum height of the terrain, black as no heights and the greys as varying levels of height. Since the car only relies on distance sensors the walls need to be tall to be detected. Although we think that the heights that we end up giving was a little too much.

Since RAW format is not supported by our word processor, we are unable to show our heightmaps but some example heightmaps are given below which were taken from the internet.

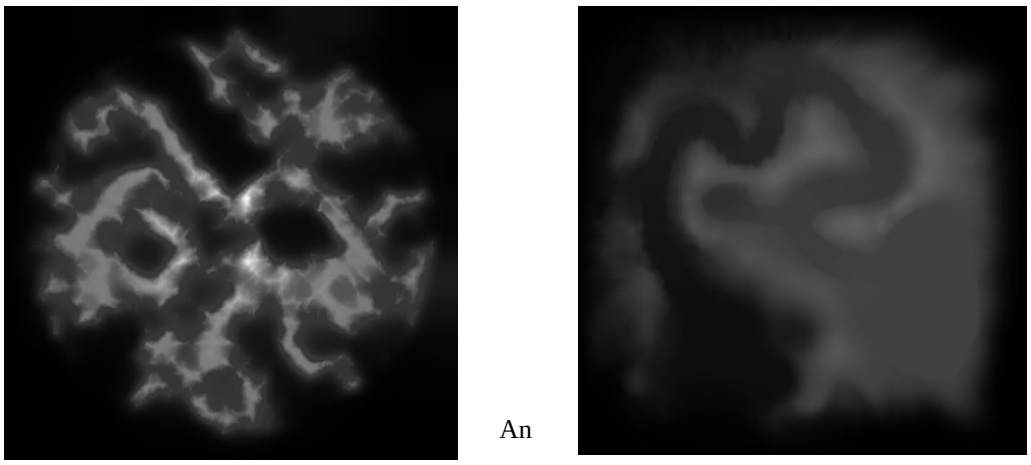


Fig: An Example heightmap for an island

Note : The heightmaps that we made and used are simple white loops on a black background to reduce the amount of time spent on it, to be minimized.

4.2. A look at the scripts

In this section we take a deeper look at the scripts that's used in this project. Before we start, we'll like to note we won't be explaining the library scripts that come with unity ml-agents package as these being in beta release, are not properly documented yet.

4.2.1 VehicleControl_Script

Now let's get into these scripts and start with the one that is very essential in our project, the VehicleControl_Script. This script as the name suggests is the scripts that has the controls for operating a car. This script is manipulated by the agent script to move the vehicle.

The VehicleControl_Script has a class known as "Wheels" that has three members, a "WheelColider object", and two Boolean variables for steerable and has motor function.

```
[System.Serializable]
public class Wheels {

    public WheelCollider wheel;
    public bool isSteerable = false;
    public bool isMotor = false;

}
```

These variables are used to hold a wheel collider object which is basically an object in unity that is used create wheels for vehicles and hence come with a ton of properties and methods such as suspension, wheel radius and steering angle.

Now we use this class in two methods known as "carMove" and "carSteer" appropriately named as they are the methods that move and steer the car respectively, Snippets of both are given below.

```
public void carMove(float moveAxis)
{
  motor = maxTorque * moveAxis; // has to be within -1 & 1

    foreach (Wheels w in wheels)
    {
        if (w.isMotor)
            w.wheel.motorTorque = motor;
    }
    As.pitch = 0.75f + (0.5f * Mathf.Abs(moveAxis));
}
```

In this method, we calculate the amount of torque to be given to the motorable wheels and assign these calculated values to the motorTorque property of the wheel collider.

The carSteer method is also written in a similar fashion with the only difference being that, in carSteer the steering angle is calculated and applied to wheels that are steerable.

```
public void carSteer(float steerAxis)
{
    steer = maxSteeringAngle * steerAxis; // has to be within -1 & 1

    foreach (Wheels w in wheels)
    {
        if (w.isSteerable)
            w.wheel.steerAngle = steer;

    }

}
```

Note: These methods are written as public so that other scripts can access these methods.

4.2.2 Sensor_Script

Now lets look at another crucial script that has three objectives. These are

- Check for collisions occurred at the 5 locations in the vehicle (Front Centre, Front Right, Front Left, Middle Left and Middle Right).
- Check if any collisions occurred at the back (Just a precautionary measure)
- And provide distance data of the sides the sensors are equipped.

And these objectives are met using three simple methods, these methods are "onCollisionEnter" "Update" and "get Distances".

```
private void OnCollisionEnter(Collision collision)
{
    if (Agent)
        Agent.hasHit();
}
```

The OnCollisionEnter method is the simplest one among them and fulfils the seconds objective. It waits until a collision is detected by unity game engine and when it does, it calls the "hasHit" method defined in the "carAgent" script.

```
void Update () {
    foreach (GameObject sensor in sensors)
    {
        Debug.DrawLine(sensor.transform.position, sensor.transform.forward * maxRange + sensor.transform.position);
        RaycastHit Hit;
        if(Physics.Raycast(sensor.transform.position, sensor.transform.forward,out Hit,maxRange))
        {

            if (Hit.distance <= thresholdDistance)
            {
                Debug.Log("Hit" + sensor.name);
                if (Agent)
                    Agent.hasHit();
            }
        }
    }
}
```

Now the Update method is what does collision detection in front using the sensors, the update function is called at every frame (that is if the application runs at 60 frames a second, then update will be called 60 times in a second) and here we use RayCast to draw an invisible line of length 10 units to the direction the sensors are facing and if the line intersects any game object with a collider the details of this object is recorded to RayCastOut object known as hit, here we retrieve the distances and check if it's less than the threshold distance (minimum distance before it is considered as a collision).

```
public List<float> getDistances()
{
    List<float> distances = new List<float>();
    foreach (GameObject sensor in sensors)
    {
        RaycastHit Hit;
        if (Physics.Raycast(sensor.transform.position, sensor.transform.forward, out Hit, maxRange))
        {
            //Debug.Log("Hit");
            distances.Add(Hit.distance);
        }
        else
            distances.Add(maxRange);

    }
    return distances;
}
```

And finally the getDistances, this method again uses Raycasts to retrieve distances but instead of checking if collision has occurred, it simply puts these distances into a list of floats and returns them.

4.2.3 carAgent Script

This is the script that is directly affects the training of the vehicle, and unlike the other scripts mentioned here that inherits from MonoBehaviour class, inherits from Agent class which is a ml-agent package content which otherwise is not included. This has methods and properties that is used to connect the unity application to the TensorFlow neural network.

```
public class SensorScript : MonoBehaviour {
```

```
public class VehicleControl_Script : MonoBehaviour {
```

```
public class carAgent : Agent
```

This script has four methods, which are “AgentReset”, “CollectObservations”, “AgentAction”, All these methods are self-explanatory, but we’ll still go through each of them, let’s start with AgentReset.

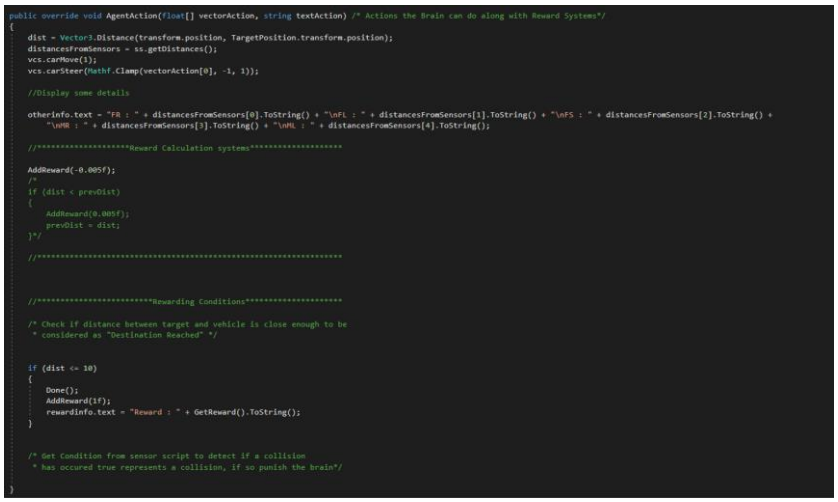
AgentReset as its name suggests executes when the agent has been reset that is when Done method is called, which usually is done when some state has been reached during training. In our case it is when a collision has occurred or the vehicle has reached the target position. In our case we reset the vehicle’s velocity, rotation and position to its original values.

```
public override void AgentReset() /* After Reseting the Agent Execute and reset some values */
{
    distancesFromSensors = ss.getDistances();
    Vector3 normalisedDistance = (TargetPosition.transform.position - transform.position).normalized;
    //dotProd = Vector3.Dot(normalisedDistance, transform.forward);
    //*************Reset velocity and position**************************
    rgbd.angularVelocity = Vector3.zero;
    rgbd.velocity = Vector3.zero;
    transform.position = StartPos;
    transform.rotation = new Quaternion(0, 0, 0, 0);
    //******************************************************************

    dist = Vector3.Distance(transform.position, TargetPosition.transform.position);
    prevDist = dist;
    //prevdotProd = dotProd;
}
```

CollectObservations method is where we give the variables that is considered as observations for the neural network. It is these variables the brain looks into to understand what's happening around it during the runtime (training or otherwise). And hence it is crucial to give relevant information here, as too many or too less can cause behaviors by the brain that are unexpected. This code block can only be written using trial and error as differentiating between major observations and minor observations for decision making is something that cannot be formulated a minor observation in a scenario may be a major one in another. In our case we give our position relative to the target and the distances that we found out from the sensor script.

```
public override void CollectObservations() /* Add the Obvs that the brain can use for decision making */
{
    Vector3 relative = TargetPosition.transform.position - transform.position;
    distancesFromSensors = ss.getDistances();

    AddVectorObs(relative.x);
    AddVectorObs(relative.z);

    AddVectorObs(rgbd.velocity.x);
    AddVectorObs(rgbd.velocity.z);

    AddVectorObs(distancesFromSensors[0]);
    AddVectorObs(distancesFromSensors[1]);
    AddVectorObs(distancesFromSensors[2]);
    AddVectorObs(distancesFromSensors[3]);
    AddVectorObs(distancesFromSensors[4]);


}
```

AgentAction is where actions that are decided by the brain is applied, the image given may be un-readable as the entire script is too big to fit in one image, but all it does is calculate the rewards and checks if the vehicle has reached its destination and reward accordingly.



```
public override void AgentAction(float[] vectorAction, string textAction) /* Actions the Brain can do along with Reward Systems*/
{
    dist = Vector3.Distance(transform.position, TargetPosition.transform.position);
    distancesFromSensors = ss.getDistances();
    vcs.carMove(1);
    vcs.carSteer(Mathf.Clamp(vectorAction[0], -1, 1));

    //Display some details

    otherinfo.text = "FR : " + distancesFromSensors[0].ToString() + "\nFL : " + distancesFromSensors[1].ToString() + "\nFS : " + distancesFromSensors[2].ToString() +
        "\nMR : " + distancesFromSensors[3].ToString() + "\nML : " + distancesFromSensors[4].ToString();

    //*******************Reward Calculation systems*******************

    AddReward(-0.005f);
    /*
    if (dist < prevDist)
    {
        AddReward(0.005f);
        prevDist = dist;
    }*/

    //****************************************************************



    //************************Rewarding Conditions*********************

    /* Check if distance between target and vehicle is close enough to be
     * considered as "Destination Reached" */

    if (dist <= 10)
    {
        Done();
        AddReward(1f);
        rewardinfo.text = "Reward : " + GetReward().ToString();
    }


    /* Get Condition from sensor script to detect if a collision
     * has occured true represents a collision, if so punish the brain*/
}
```

```
public void hasHit()
{
    Done();
    AddReward(-1f);
    rewardinfo.text = "Reward : " + GetReward().ToString();
}
```

The final method is hasHit which simple resets the agent and rewards the brain an negative reward, as it is only called when a collision has occurred.

4. 3 Training Results

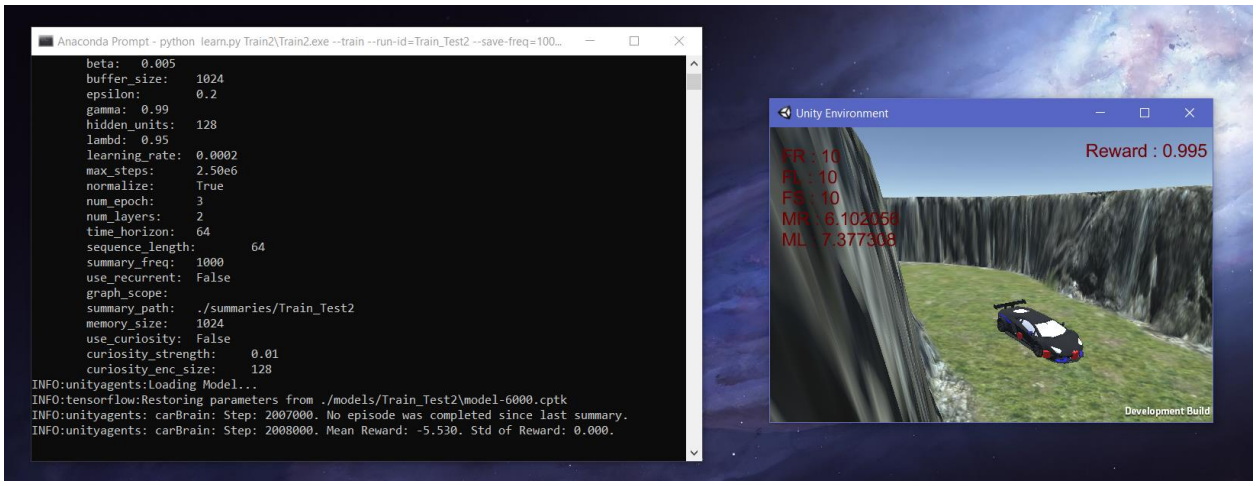


Fig 1: Vehicle being trained, process continuing from 2007000 iteration onwards

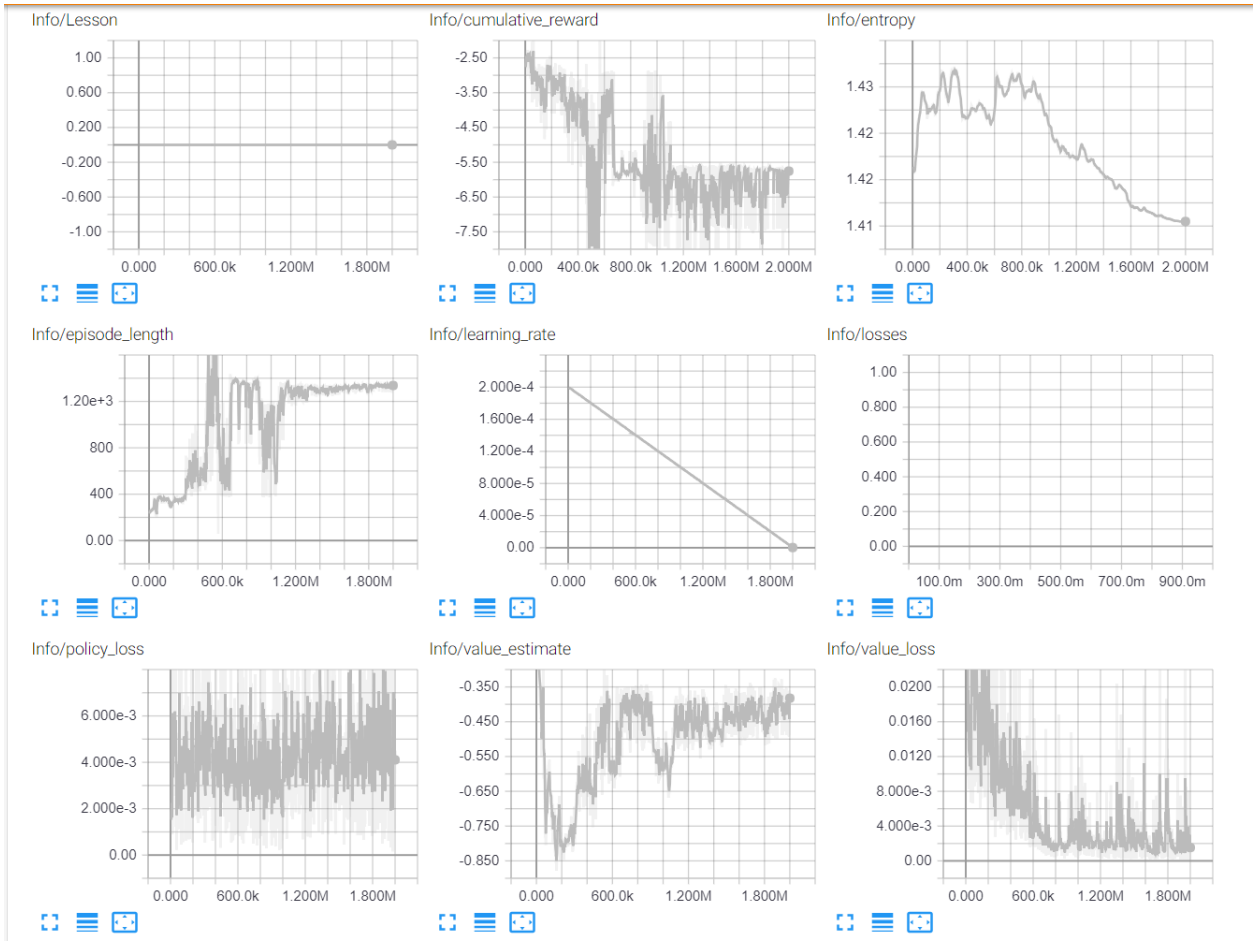


Fig 2: TensorBoard displaying details about the current training

5. Project Results

After training was finished and the Bytes file produced was assigned to the brain and after adding some UI, the project was built as a standalone binary application (.exe file). The output of the following is given below;

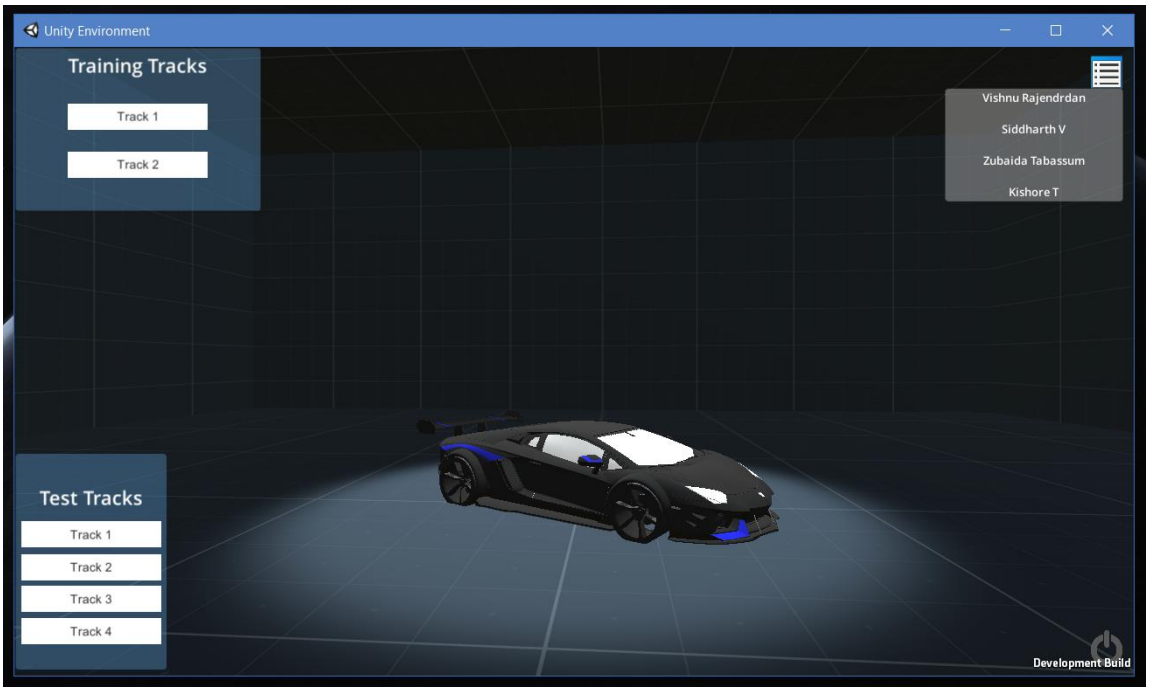


Fig : Main Menu of the final build

Fig: First of two Training Tracks

Fig : One of four Test Tracks

6. CONCLUSION

Using Unity, ML-Agents and C# scripting we were able to build a prototype of a self-driving car with distance sensors. Due to the ML-Agents limited capability because of its beta phase release, a more capable system with camera setups cannot be built. Nonetheless our project becomes a proof of concept of teaching the Vehicle in a virtual world where all the variables can controlled. At a later stage where ML-Agents become more efficient and capable, a fully featured simulation can be created with traffic ,dynamic weather and pedestrian activity. As machine learning is still new, for now having a ML assisted Heuristic system is better suited for real world applications of the concept of the project, as unpredictable and rare occasions are something that even though can be avoided will be tough to train for.

Large companies such as Google and Tesla know this as a fact and therefore made it a rule for all driver less vehicles to have a human personal present at all time to solve this crisis,as quick decision making skill that is gifted to humans and which is not reproduceable with our current technology until the point of singularity is achieved (True AI), is essential in the real world to avoid some of the accidents.

7. REFERENCES

1. Unity User Reference Manual
   url: https://docs.unity3d.com/Manual/index.html

2. ML-Agents GitHub repository:
   url: https://github.com/Unity-Technologies/ml-agents

3. medium.com for ML-Agents Installation :
   url: https://medium.com/@indiecontessa/setting-up-a-python-environment-with-tensorflow-on-macos-for-training-unity-ml-agents-faf19d71201