G-One Debuggers

Moway

The Final Report

For Virtual self-driving car using reinforcement learning

Version 1.0

V.Sajeevan

160544C

24th May, 2019

Abstract

Moway is a computational framework, built to support a key need triggered by the rapid growth of autonomy in-ground map along with manned vehicles. a virtual self-driving car using reinforcement learning to mimic the behavior when it is running with a set of another self-driving car or human cars. This can be used to observe the impact that the self-driving car can have on damping the traffic bottlenecks caused by manned vehicles. This framework can be simulatable in a different traffic situation and different phenomena. There is more type of autonomous vehicles in this new technology. But there are only a few simulating frameworks to test those autonomous vehicles and lack of tools to simulate different traffic phenomena in the presence of autonomous cars.

The application consists of four main functionalities. They are Training, Playing, Store and Load Data and Create Environment. The ultimate goal of all these functionalities is Learning from the input in the designed virtual environment and drive autonomously and able to detect the obstacles (2D environment pixels) in the pathway of the virtual environment and travel throw the path safely.

The development of the project has been carried out under the Rational Unified Process methodology with several iterations. The architecture of the application follows a layered architecture which consists of a View layer, a Business logic layer, and a Data access layer. Software engineering good practices have been adopted in the development process for scheduling, requirement specification, coding, and testing. Python had been used as languages for coding. Model View ViewModel design pattern which is specialized for the pygame library had also been used. Version controlling had been carried out using GitHub. Testing had been carried out with a number of the framework from Python. The application can be downloaded as a desktop version, which is the advantage granted through universal apps. This report starts with an introduction. Then it describes the literature review, system models, system implementation, system testing and analysis in preceding sections. Conclusion and future work are described in the last section.

Table of Contents

Final Report

1. Introduction

1.1. Background of the application domain/ problem

Moway 1.0 is Desktop base virtual self-driving simulation framework. The framework is about designing and developing an environment along with virtual self-driving car using reinforcement learning to mimic the behavior when it is running with a set of another self-driving car or human cars. This can be used to observe the impact that the self-driving car can have on damping the traffic bottlenecks caused by manned vehicles. This framework can be simulatable in a different traffic situation and different phenomena.

This product is a desktop offline framework it can simulatable an autonomous vehicle in different phenomena. The autonomous vehicle should be able to analyze the environment and store the data in local storage. Able to give predictions about the environment. The environment should be able to change the user's likeness

1.2. The motivation for the selected system development

During the proposal phase of this project, I did a search on similar applications. The research revealed that there are many similar applications in the market, There is more type of autonomous vehicles in this new technology. But there are only a few simulating frameworks to test those autonomous vehicles and lack of tools to simulate different traffic phenomena in the presence of autonomous cars.

1.3. Importance and the main purpose of the system

The ultimate objective of this framework provides a user-friendly and easy way of creating a new environment and simulating Training and Playing in a virtual environment. And also There is more type of autonomous vehicles in this new technology. But there are only a few simulating frameworks to test those autonomous vehicles and lack of tools to simulate different traffic phenomena in the presence of autonomous cars.

1.4. Overview of the system used approach and outcome

Moway 1.0 is Desktop base virtual self-driving simulation framework. The framework is about desig ning and developing an environment along with virtual self-driving car using reinforcement learning to mimic the behavior when it is running with a set of another self-driving car or human cars. This can be used to observe the impact that the self-driving car can have on damping the traffic bottlenecks caused by manned vehicles. This framework can be simulatable in a different traffic situation and different phenomena.

This framework provides a user-friendly and easy way of creating a new environment and simulating Training and Playing in a virtual environment.

2. Literature Review

2.1. The Key Idea

With great development in technology, life is also getting busier day by day. The traffic is the major problem in Sri Lankan roads. We lose our lot of valuable time in the traffic. This product is a desktop offline framework it can simulatable an autonomous vehicle in different phenomena. The autonomous vehicle should be able to analyze the environment and store the data in local storage. Able to give predictions about the environment. The environment should be able to change the user's likeness.

During the proposal phase of this project, I did a search on similar applications. The research revealed that there are many similar applications in the market, There is more type of autonomous vehicles in this new technology. But there are only a few simulating frameworks to test those autonomous vehicles and lack of tools to simulate different traffic phenomena in the presence of autonomous cars.

2.2 Related Developments and Products available

There are many kinds of self-driven virtual and real car available. They use a Q-learning, neural networks, Pygame and also reinforcement learning.

- *Flow: Architecture and Benchmarking for Reinforcement Learning in Traffic Control*

  Flow is a new computational framework, built to support a key need triggered by the rapid growth of autonomy in ground traffic: controllers for autonomous vehicles in the presence of complex nonlinear dynamics in traffic. Leveraging recent advances in deep Reinforcement Learning (RL), Flow enables the use of RL methods such as policy gradient for traffic control and enables benchmarking the performance of classical (including hand-designed) controllers with learned policies (control laws). Flow integrates traffic micro simulator SUMO with deep reinforcement learning library rllab and enables the easy design of traffic tasks, including different networks configurations and vehicle dynamics.

3. Specific Requirements

This section states all the requirements required in virtual self-driving car and simulation framework. All the requirements are stated with details and they are divided into categories depending on their area

3.1 System Requirements

3.1.1 Functional Requirements

The application consists of four main functionalities. They are Training, Playing, Store and Load Data and Create Environment. Out of them, the most targeted function is Train a virtual self-driving car. In this service, the user wants to design a virtual environment Using UI. There are 3 main Sence GUIs. Main and more functionalities are in the Create Environment Menu GUI. User can Change The map using Right and Left Arrow Keys, And also user can Change the adding obstacle or manned car using Up and Down Arrow Keys. User can set the location of the obstacle by using mouse Click button and also set the angle of obstacle by releasing the mouse button.

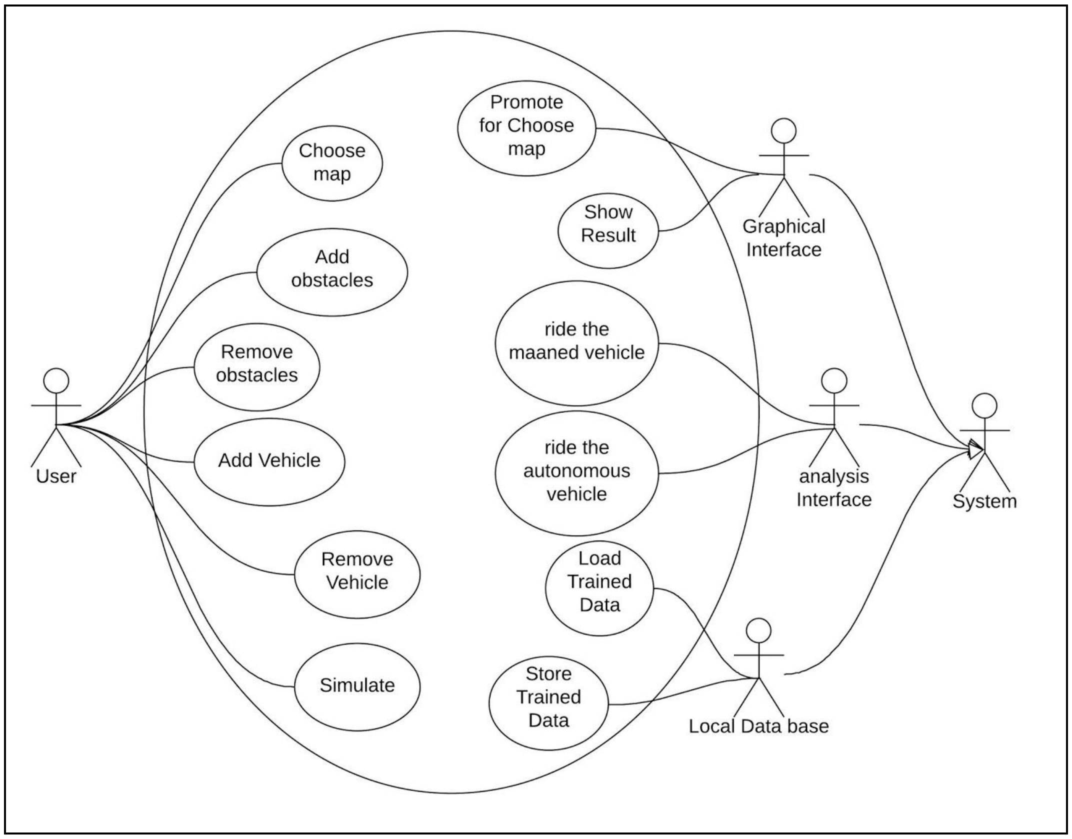


Figure 3.1.1 - Use Case Diagram.

3.1.2 Non Functional Requirements

A number of nonfunctional requirements have been figured out during the requirement gathering phase. They can be classified according to usability, reliability, performance, accessibility, and supportability.

3.1.2.1 Usability

Training time for the users - This application should not be very complicated. Usually, autonomous systems need high time duration to train. In this framework, the user can save the trained data in local storage so the user does not want to re-train the vehicle.

Friendly and attractive user interface - Users of the system should be able to interact with the system easily and the interaction should be easily understandable without any uncertainties.

Easy to use - The users will not need any extra skills or high technical knowledge to use the system. Basic knowledge about using the computer will be enough to use the system.

3.1.2.2 Reliability, availability and Accuracy

This framework is an offline desktop framework.it also available at any time for the user unless there are any updates or problems.

Automated systems analyzing and forecasting should be very accurate. Forecasting error should be minimum. The forecasted error should be varied within a 5% range from the actual value. The system will not lose any trained data files from the framework and will make sure that the files are saved in local storage.

3.1.2.3 Performance

The framework should be user-friendly, appropriate error messages should be displayed and the system should be handling several inputs simultaneously. The system is expected to be responsive all the time with the minimum response time.

Since the training through the dataset is computationally exhaustive at times the training gets interrupted and hence it slows down the development of the system. So to mitigate it exploit the different system to develop and simulate.

3.1.2.4 Supportability

The naming of the classes and the tables of the database is expected to be simple and make sense. This is because there is the possibility of the security measures being changed with time and the system will have to be changed accordingly as well. The code is expected to be commented where necessary thus facilitating easy understanding. The program logic is expected to be simple to avoid unnecessary complexities arriving at any failure.

3.2 System Design

3.2.1 System Architecture

The system follows a layered architecture. The three-tier architecture used is shown in figure 3.2.1. The architecture has three layers, a user interface layer (view layer) on top, a business logic layer in the middle, and a data storage layer at the bottom. Each layer exposes an interface (API) to be used by the above layers. Advantages of this architecture are increasing abstraction levels, standardization of layer interfaces for libraries, frameworks and design patterns used.

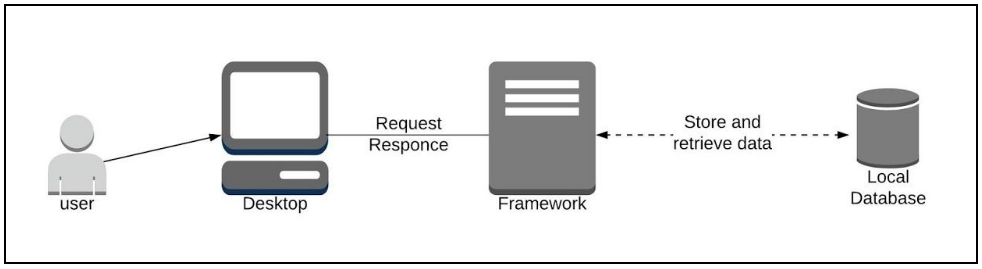


Figure 3.2.1- Layered Architecture

The View layer consists of the interfaces with which the user interacts with the application. Logical Layer (Domain Layer) contains the core of the application, where all processing and manipulating activities happen. An algorithm is implemented within the logic layer to calculate the route from one location to another location. Data Access Layer is responsible for retrieving data from the data file to calculate the route. All three layers communicate with each other to generate accurate search results to the user. For the ease of implementing the above architecture within the system, 'Model View ViewModel" design pattern is used and its intent is to provide a clean separation between the user interface controls and their logic. There are three components to it. The Model, View, and ViewModel have distinct and separate roles. At a higher level, View knows about the ViewModel and ViewModel knows about the Model. But Model is unaware of the ViewModel and the ViewModel is unaware of the View. The view is responsible for encapsulating the user interface and its logic. View model handles the presentation logic and the state while acting as the mediator between the View and the Model.

3.2.2 Logical View

The class diagram for the system is shown in figure 3.2.2. Different classes belong to different components under Model, View, and ViewModel. The logical view gives an overall representation of the Moway system without going into detailed architectural levels. The logical view is the object model of the design which concerned with the functionality that the system provides to end-users. UML Diagrams used to represent the logical view include Class diagram, Communication diagram, and Sequence diagram.

The View consists of the classes Environment and Menu and EnvironmentMenu. These classes mainly handle initializing the user interface components and click events. The Model includes the Car, RLCar, RuleBasedCar, Obstacle, and map classes. ViewModel includes two classes Training and Playing Algorithm.

The overview of the Moway system can be shown using a Class diagram as follows.

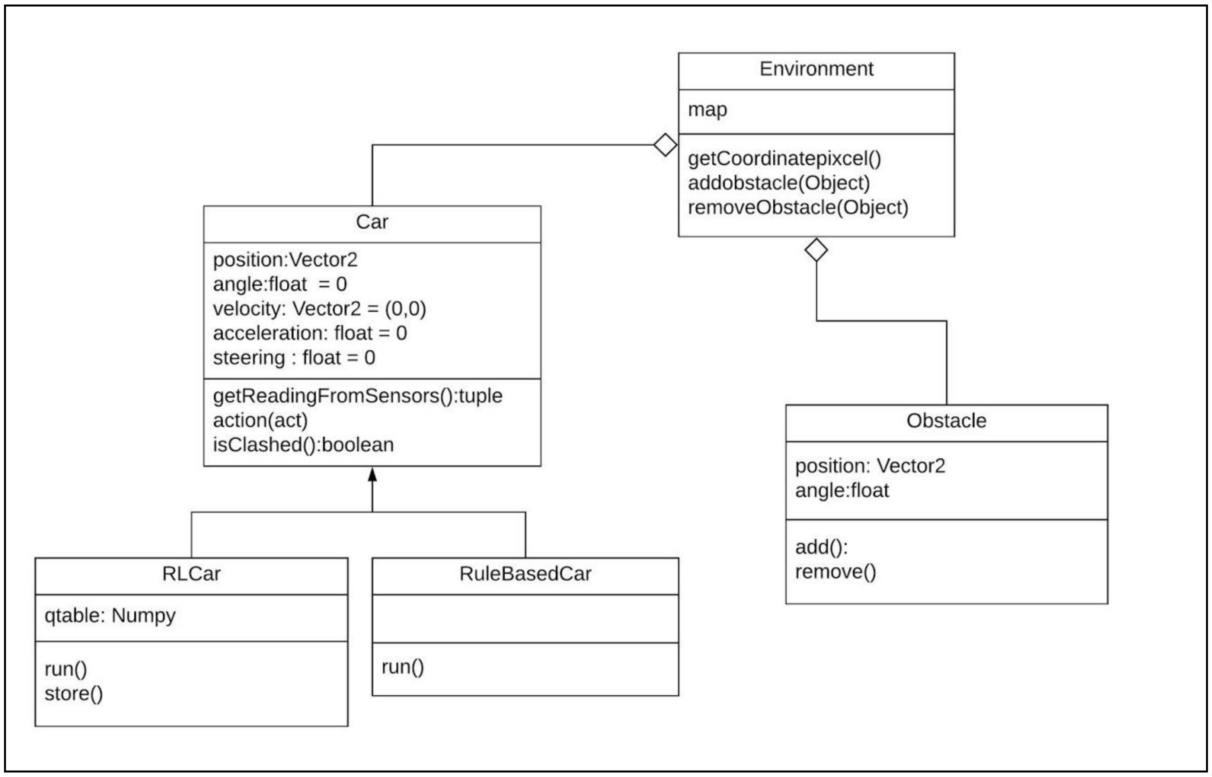


Figure 3.2.2 - Class Diagram

3.2.3 Process View

The process view illustrates how the system behaves at runtime by explaining the system processes and how they communicate dynamically. Following will be the most important processes that have a high impact on the system architecture.

The sequence diagrams for the system shows how processes operate with one another in what order. It specifies object interactions arranged in time sequence. The sequence diagram for the Start to Destination function is shown in figure 3.2.3.

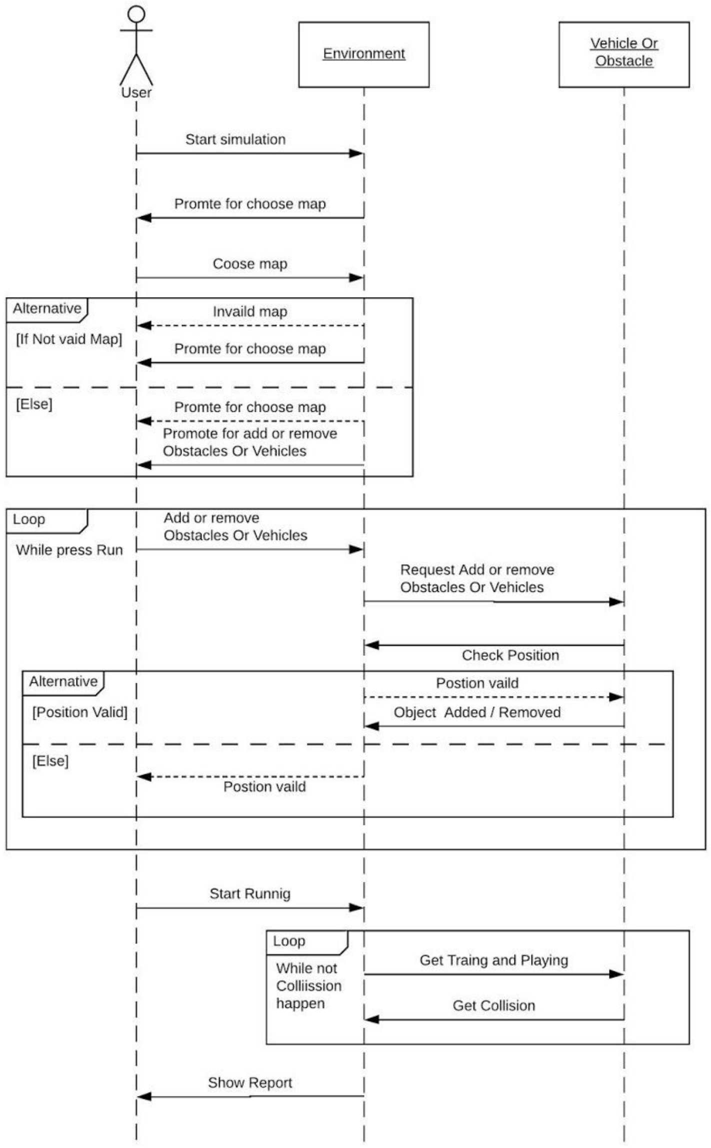


Figure 5 - Sequencel diagram

3.3 Database Design

The Moway framework will be using Numpy python library database as the centralized data storage for store autonomous vehicle's trained data. It is a free and open-source document-oriented library. Those data will store on a local disk. The user can Load the trained data for simulating the same system.

4. System Implementation

4.1 Implementation Procedure

4.1.1 Target Platform

This product is a desktop offline framework it can simulatable an autonomous vehicle in different phenomena. The autonomous vehicle should be able to analyze the environment and store the data in local storage. Able to give predictions about the environment. The environment should be able to change the user's likeness. As this is a desktop based application, the users can easily use the system by entering the computer or laptop. They will not require an internet connection to access the framework.

4.1.2 Technologies and tools

There is more type of autonomous vehicles in this new technology. But there are only a few simulating frameworks to test those autonomous vehicles and lack of tools to simulate different traffic phenomena in the presence of autonomous cars.

The perspective of this product makes the testing of autonomous vehicles easier in virtual. This framework can be simulatable in different traffic situations like driving on the straight road, curved road, junction, and traffic with manned vehicles. Users can change the map path and they can add the obstacles and manned vehicles along with the virtual self-driving car.

The system will be using Python Numpy library as the database. JetBrains PyCharm IDE will be used to develop the system.

4.1.3 Approach

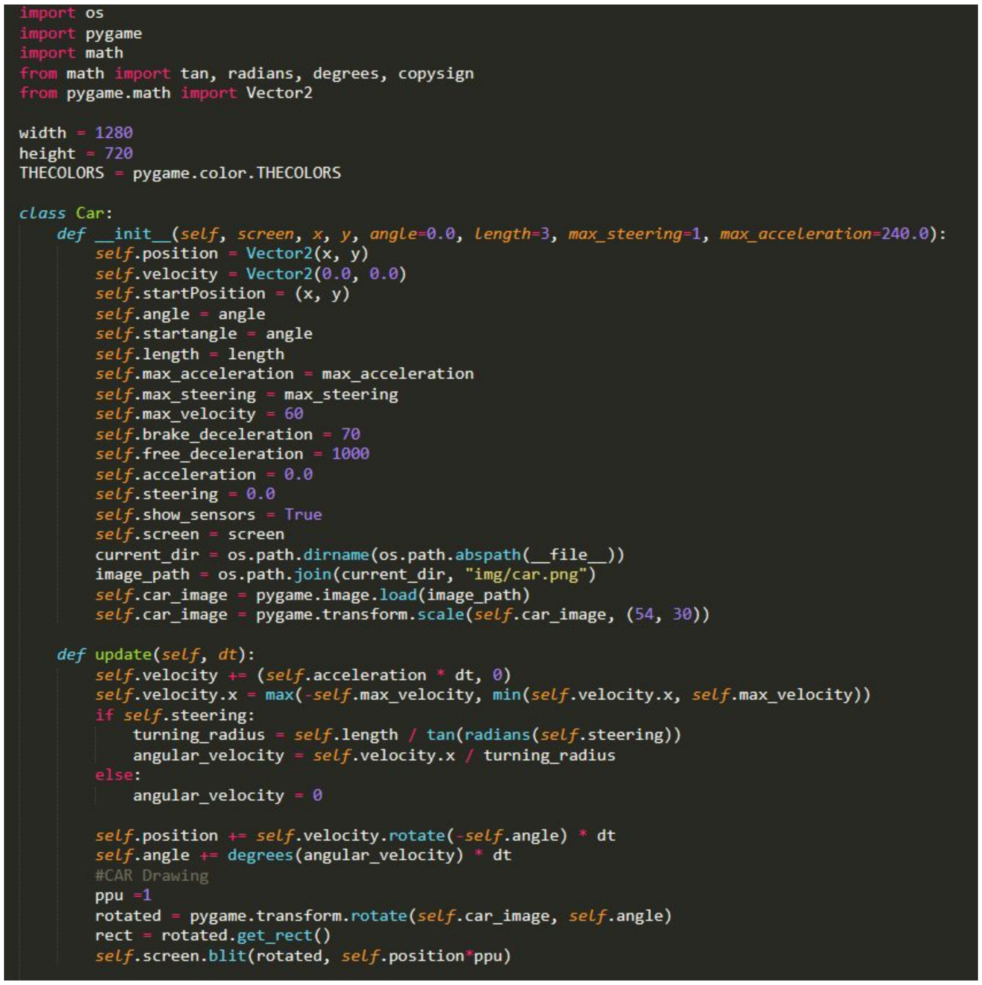
JetBrains PyCharm Community Edition with Windows SDK has been used as the IDE. The software was basically coded with python. User interfaces had been coded using pygame library. Model View ViewModel design pattern had been used to structure the project. The project Folder Consists there main folders. They are Model Classes, Images, Saved model. The application software was developed according to the Model View ViewModel. Therefore requirement analysis, designing, coding and testing were done in parallel.

4.2 Materials

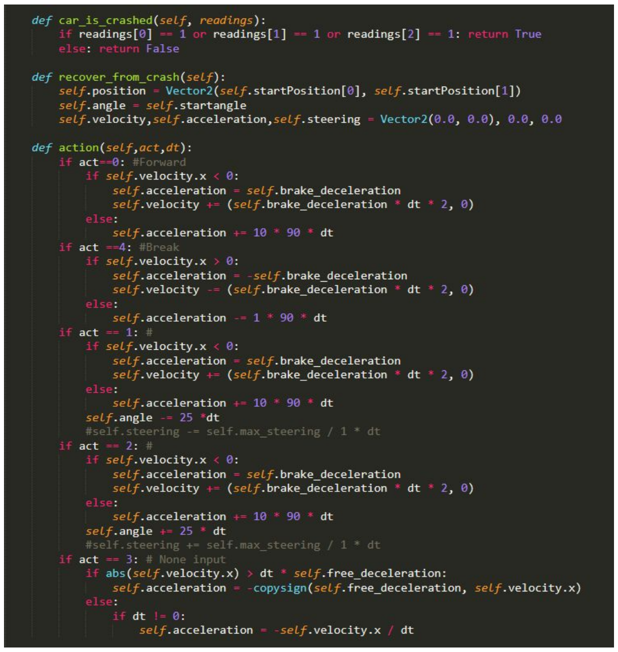
When implementing the system some existing libraries were used to increase reusability. One such library was the Apache Commons Math to calculate the correlation and variance among the GUI position. When implementing the user interfaces, The library called Pygame was employed. Using this, it was easy to improve the aesthetic aspects of the user interfaces. The results analysis subsystem heavily uses graphs to plot relationships, analysed data etc. These Reinforcement learning analysis using an external library called “keras” and also Using TensorFlow backend.

4.3 The algorithm

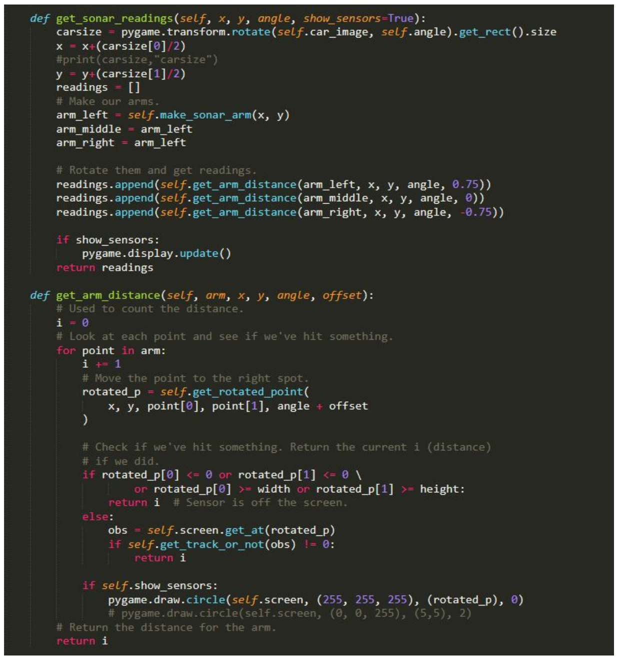
The algorithm part of the framework had been implemented to model of car is below



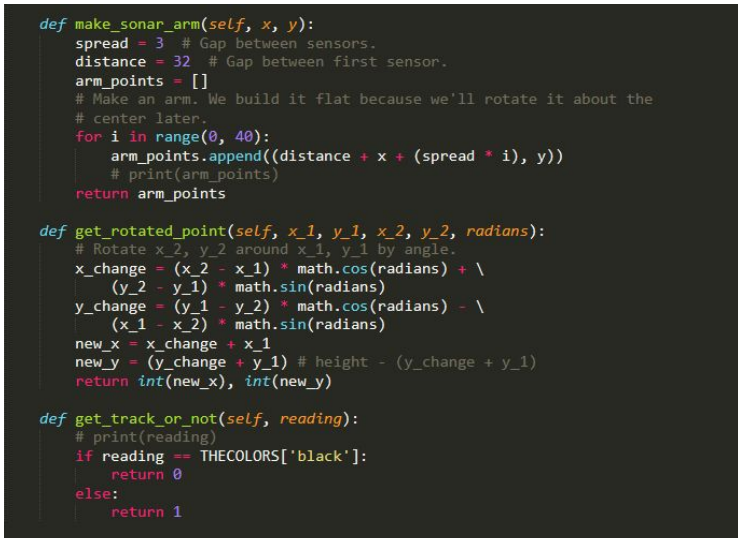
```python
import os
import pygame
import math
from math import tan, radians, degrees, copysign
from pygame.math import Vector2

width = 1280
height = 720
THECOLORS = pygame.color.THECOLORS

class Car:
    def __init__(self, screen, x, y, angle=0.0, length=3, max_steering=1, max_acceleration=240.0):
        self.position = Vector2(x, y)
        self.velocity = Vector2(0.0, 0.0)
        self.startPosition = (x, y)
        self.angle = angle
        self.startangle = angle
        self.length = length
        self.max_acceleration = max_acceleration
        self.max_steering = max_steering
        self.max_velocity = 60
        self.brake_deceleration = 70
        self.free_deceleration = 1000
        self.acceleration = 0.0
        self.steering = 0.0
        self.show_sensors = True
        self.screen = screen
        current_dir = os.path.dirname(os.path.abspath(__file__))
        image_path = os.path.join(current_dir, "img/car.png")
        self.car_image = pygame.image.load(image_path)
        self.car_image = pygame.transform.scale(self.car_image, (54, 30))

    def update(self, dt):
        self.velocity += (self.acceleration * dt, 0)
        self.velocity.x = max(-self.max_velocity, min(self.velocity.x, self.max_velocity))
        if self.steering:
            turning_radius = self.length / tan(radians(self.steering))
            angular_velocity = self.velocity.x / turning_radius
        else:
            angular_velocity = 0

        self.position += self.velocity.rotate(-self.angle) * dt
        self.angle += degrees(angular_velocity) * dt
        #CAR Drawing
        ppu =1
        rotated = pygame.transform.rotate(self.car_image, self.angle)
        rect = rotated.get_rect()
        self.screen.blit(rotated, self.position*ppu)
```



```python
def car_is_crashed(self, readings):
    if readings[0] == 1 or readings[1] == 1 or readings[2] == 1: return True
    else: return False

def recover_from_crash(self):
    self.position = Vector2(self.startPosition[0], self.startPosition[1])
    self.angle = self.startangle
    self.velocity,self.acceleration,self.steering = Vector2(0.0, 0.0), 0.0, 0.0

def action(self,act,dt):
    if act==0: #Forward
        if self.velocity.x < 0:
            self.acceleration = self.brake_deceleration
            self.velocity += (self.brake_deceleration * dt * 2, 0)
        else:
            self.acceleration += 10 * 90 * dt
    if act ==4: #Break
        if self.velocity.x > 0:
            self.acceleration = -self.brake_deceleration
            self.velocity -= (self.brake_deceleration * dt * 2, 0)
        else:
            self.acceleration -= 1 * 90 * dt
    if act == 1: #
        if self.velocity.x < 0:
            self.acceleration = self.brake_deceleration
            self.velocity += (self.brake_deceleration * dt * 2, 0)
        else:
            self.acceleration += 10 * 90 * dt
        self.angle -= 25 *dt
        #self.steering -= self.max_steering / 1 * dt
    if act == 2: #
        if self.velocity.x < 0:
            self.acceleration = self.brake_deceleration
            self.velocity += (self.brake_deceleration * dt * 2, 0)
        else:
            self.acceleration += 10 * 90 * dt
        self.angle += 25 * dt
        #self.steering += self.max_steering / 1 * dt
    if act == 3: # None input
        if abs(self.velocity.x) > dt * self.free_deceleration:
            self.acceleration = -copysign(self.free_deceleration, self.velocity.x)
        else:
            if dt != 0:
                self.acceleration = -self.velocity.x / dt
```

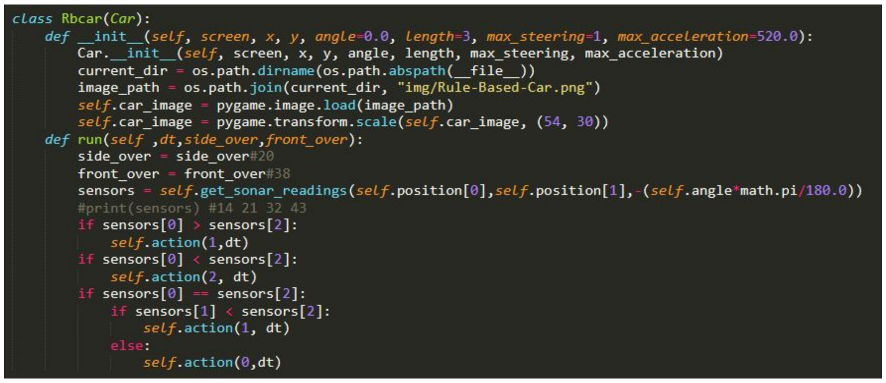


```python
def get_sonar_readings(self, x, y, angle, show_sensors=True):
    carsize = pygame.transform.rotate(self.car_image, self.angle).get_rect().size
    x = x+(carsize[0]/2)
    #print(carsize,"carsize")
    y = y+(carsize[1]/2)
    readings = []
    # Make our arms.
    arm_left = self.make_sonar_arm(x, y)
    arm_middle = arm_left
    arm_right = arm_left

    # Rotate them and get readings.
    readings.append(self.get_arm_distance(arm_left, x, y, angle, 0.75))
    readings.append(self.get_arm_distance(arm_middle, x, y, angle, 0))
    readings.append(self.get_arm_distance(arm_right, x, y, angle, -0.75))

    if show_sensors:
        pygame.display.update()
    return readings

def get_arm_distance(self, arm, x, y, angle, offset):
    # Used to count the distance.
    i = 0
    # Look at each point and see if we've hit something.
    for point in arm:
        i += 1
        # Move the point to the right spot.
        rotated_p = self.get_rotated_point(
            x, y, point[0], point[1], angle + offset
        )

        # Check if we've hit something. Return the current i (distance)
        # if we did.
        if rotated_p[0] <= 0 or rotated_p[1] <= 0 \
                or rotated_p[0] >= width or rotated_p[1] >= height:
            return i  # Sensor is off the screen.
        else:
            obs = self.screen.get_at(rotated_p)
            if self.get_track_or_not(obs) != 0:
                return i

        if self.show_sensors:
            pygame.draw.circle(self.screen, (255, 255, 255), (rotated_p), 0)
            # pygame.draw.circle(self.screen, (0, 0, 255), (5,5), 2)
    # Return the distance for the arm.
    return i
```



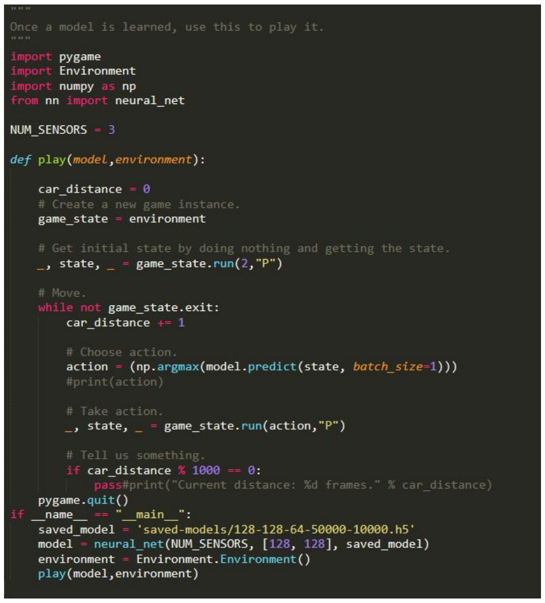
```python
def make_sonar_arm(self, x, y):
    spread = 3  # Gap between sensors.
    distance = 32  # Gap between first sensor.
    arm_points = []
    # Make an arm. We build it flat because we'll rotate it about the
    # center later.
    for i in range(0, 40):
        arm_points.append((distance + x + (spread * i), y))
        # print(arm_points)
    return arm_points

def get_rotated_point(self, x_1, y_1, x_2, y_2, radians):
    # Rotate x_2, y_2 around x_1, y_1 by angle.
    x_change = (x_2 - x_1) * math.cos(radians) + \
        (y_2 - y_1) * math.sin(radians)
    y_change = (y_1 - y_2) * math.cos(radians) - \
        (x_1 - x_2) * math.sin(radians)
    new_x = x_change + x_1
    new_y = (y_change + y_1) # height - (y_change + y_1)
    return int(new_x), int(new_y)

def get_track_or_not(self, reading):
    # print(reading)
    if reading == THECOLORS['black']:
        return 0
    else:
        return 1
```

The Car class is inherited as RLCar(Reinforcement Learning Car) and RBCar (Rule-based Car)

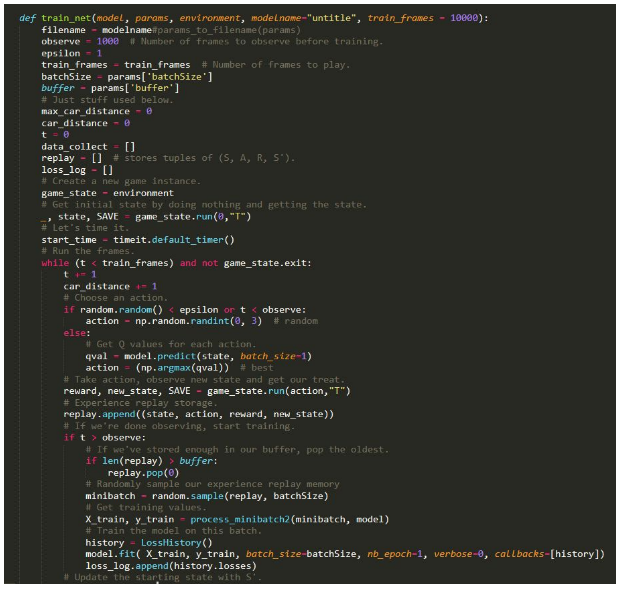


```python
class Rbcar(Car):
    def __init__(self, screen, x, y, angle=0.0, length=3, max_steering=1, max_acceleration=520.0):
        Car.__init__(self, screen, x, y, angle, length, max_steering, max_acceleration)
        current_dir = os.path.dirname(os.path.abspath(__file__))
        image_path = os.path.join(current_dir, "img/Rule-Based-Car.png")
        self.car_image = pygame.image.load(image_path)
        self.car_image = pygame.transform.scale(self.car_image, (54, 30))
    def run(self ,dt,side_over,front_over):
        side_over = side_over#20
        front_over = front_over#38
        sensors = self.get_sonar_readings(self.position[0],self.position[1],-(self.angle*math.pi/180.0))
        #print(sensors) #14 21 32 43
        if sensors[0] > sensors[2]:
            self.action(1,dt)
        if sensors[0] < sensors[2]:
            self.action(2, dt)
        if sensors[0] == sensors[2]:
            if sensors[1] < sensors[2]:
                self.action(1, dt)
            else:
                self.action(0,dt)
```

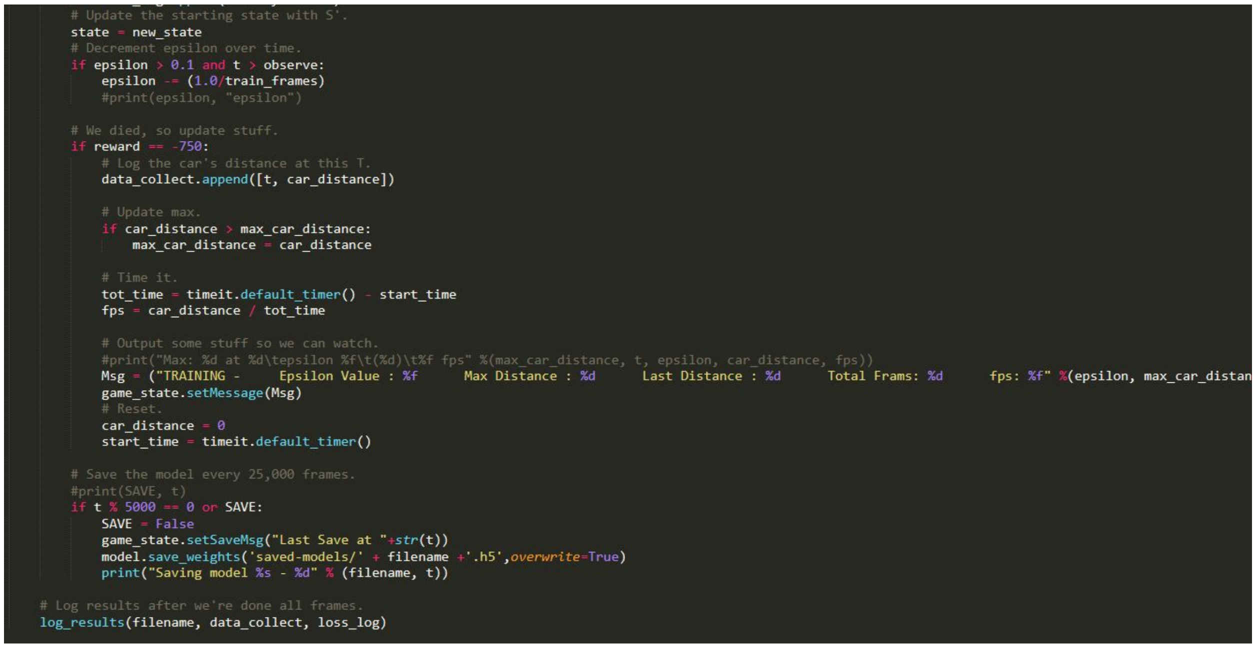
When simulating the model user able play the model and alos train the model.



```python
"""
Once a model is learned, use this to play it.
"""
import pygame
import Environment
import numpy as np
from nn import neural_net

NUM_SENSORS = 3

def play(model,environment):

    car_distance = 0
    # Create a new game instance.
    game_state = environment

    # Get initial state by doing nothing and getting the state.
    _, state, _ = game_state.run(2,"P")

    # Move.
    while not game_state.exit:
        car_distance += 1

        # Choose action.
        action = (np.argmax(model.predict(state, batch_size=1)))
        #print(action)

        # Take action.
        _, state, _ = game_state.run(action,"P")

        # Tell us something.
        if car_distance % 1000 == 0:
            pass#print("Current distance: %d frames." % car_distance)
    pygame.quit()
if __name__ == "__main__":
    saved_model = 'saved-models/128-128-64-50000-10000.h5'
    model = neural_net(NUM_SENSORS, [128, 128], saved_model)
    environment = Environment.Environment()
    play(model,environment)
```

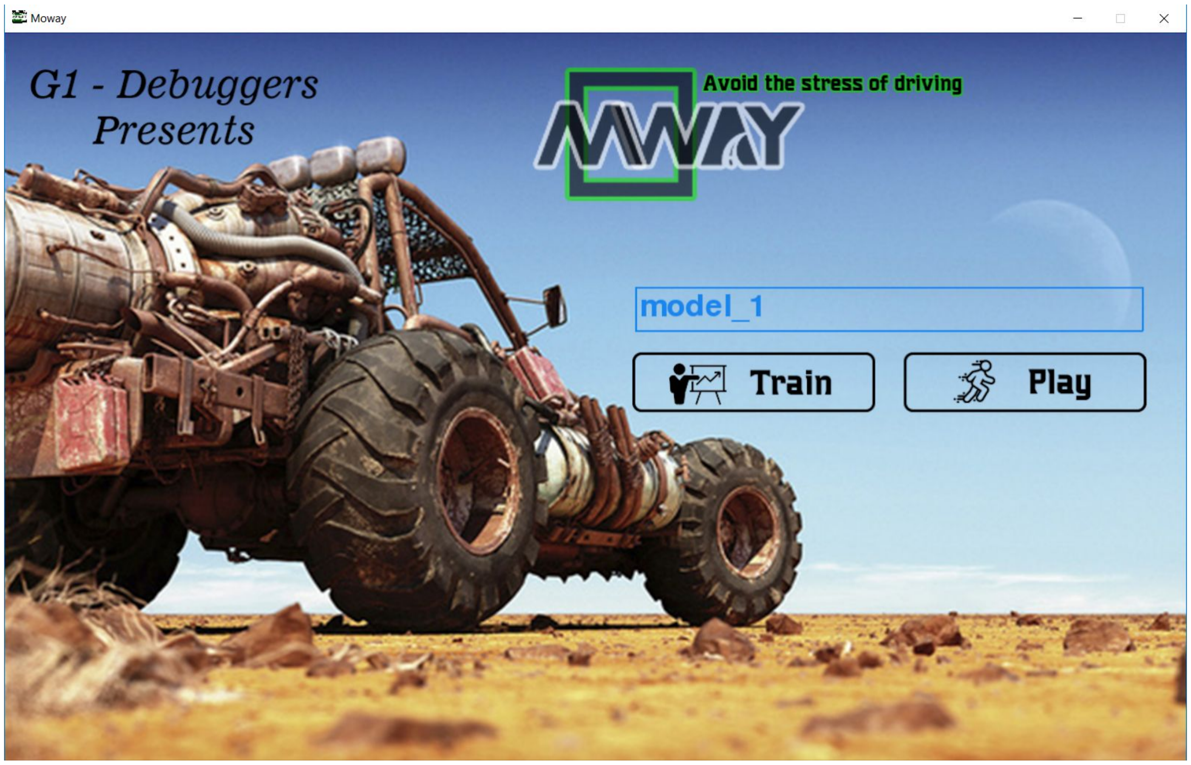


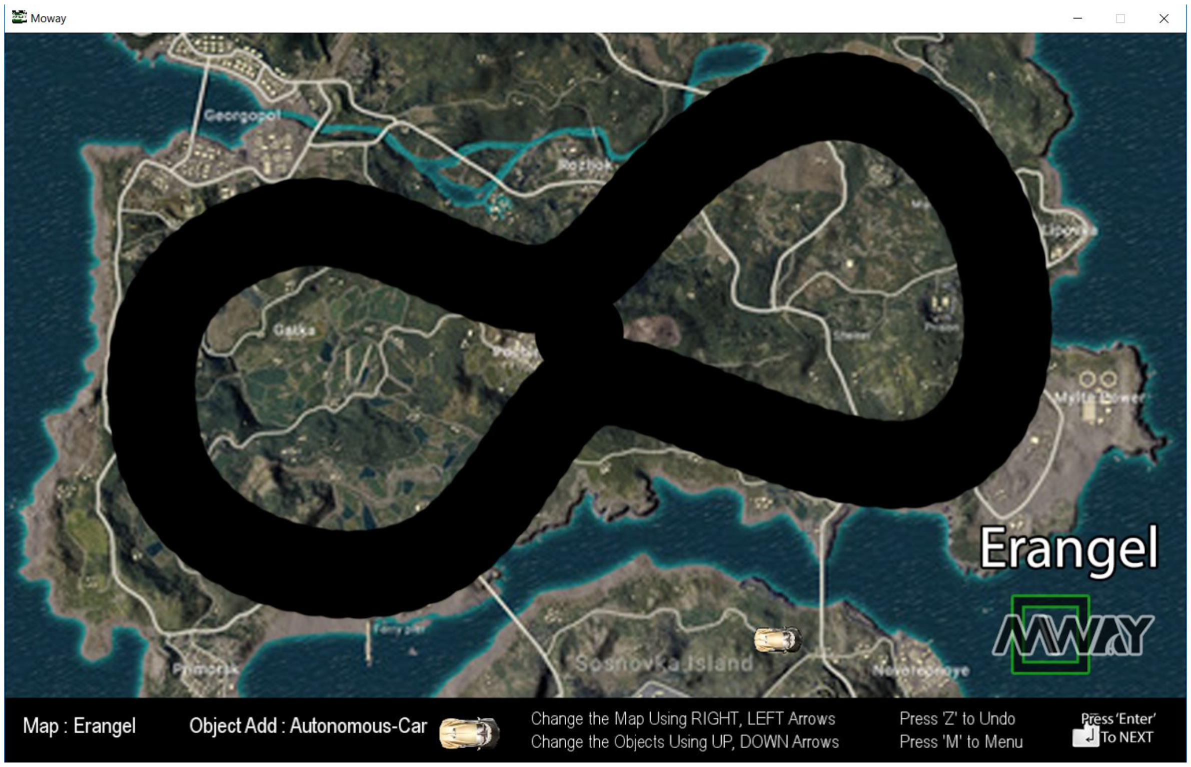
```python
def train_net(model, params, environment, modelname="untitle", train_frames = 10000):
    filename = modelname#params_to_filename(params)
    observe = 1000  # Number of frames to observe before training.
    epsilon = 1
    train_frames = train_frames  # Number of frames to play.
    batchSize = params['batchSize']
    buffer = params['buffer']
    # Just stuff used below.
    max_car_distance = 0
    car_distance = 0
    t = 0
    data_collect = []
    replay = []  # stores tuples of (S, A, R, S').
    loss_log = []
    # Create a new game instance.
    game_state = environment
    # Get initial state by doing nothing and getting the state.
    _, state, SAVE = game_state.run(0,"T")
    # Let's time it.
    start_time = timeit.default_timer()
    # Run the frames.
    while (t < train_frames) and not game_state.exit:
        t += 1
        car_distance += 1
        # Choose an action.
        if random.random() < epsilon or t < observe:
            action = np.random.randint(0, 3)  # random
        else:
            # Get Q values for each action.
            qval = model.predict(state, batch_size=1)
            action = (np.argmax(qval))  # best
        # Take action, observe new state and get our treat.
        reward, new_state, SAVE = game_state.run(action,"T")
        # Experience replay storage.
        replay.append((state, action, reward, new_state))
        # If we're done observing, start training.
        if t > observe:
            # If we've stored enough in our buffer, pop the oldest.
            if len(replay) > buffer:
                replay.pop(0)
            # Randomly sample our experience replay memory
            minibatch = random.sample(replay, batchSize)
            # Get training values.
            X_train, y_train = process_minibatch2(minibatch, model)
            # Train the model on this batch.
            history = LossHistory()
            model.fit( X_train, y_train, batch_size=batchSize, nb_epoch=1, verbose=0, callbacks=[history])
            loss_log.append(history.losses)
        # Update the starting state with S'.
```



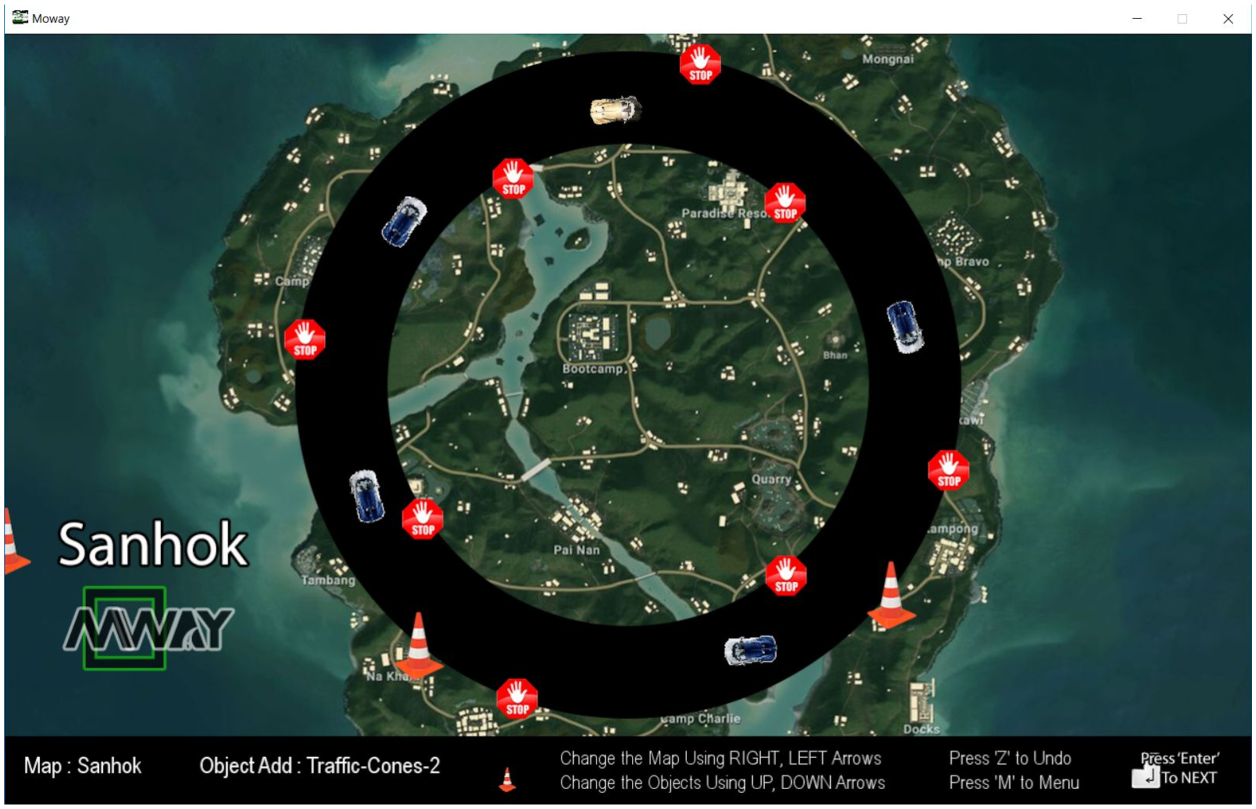
```python
        # Update the starting state with S'.
        state = new_state
        # Decrement epsilon over time.
        if epsilon > 0.1 and t > observe:
            epsilon -= (1.0/train_frames)
            #print(epsilon, "epsilon")

        # We died, so update stuff.
        if reward == -750:
            # Log the car's distance at this T.
            data_collect.append([t, car_distance])

            # Update max.
            if car_distance > max_car_distance:
                max_car_distance = car_distance

            # Time it.
            tot_time = timeit.default_timer() - start_time
            fps = car_distance / tot_time

            # Output some stuff so we can watch.
            #print("Max: %d at %d\tepsilon %f\t(%d)\t%f fps" %(max_car_distance, t, epsilon, car_distance, fps))
            Msg = ("TRAINING -      Epsilon Value : %f      Max Distance : %d      Last Distance : %d      Total Frams: %d      fps: %f" %(epsilon, max_car_distan
            game_state.setMessage(Msg)
            # Reset.
            car_distance = 0
            start_time = timeit.default_timer()

        # Save the model every 25,000 frames.
        #print(SAVE, t)
        if t % 5000 == 0 or SAVE:
            SAVE = False
            game_state.setSaveMsg("Last Save at "+str(t))
            model.save_weights('saved-models/' + filename +'.h5',overwrite=True)
            print("Saving model %s - %d" % (filename, t))

    # Log results after we're done all frames.
    log_results(filename, data_collect, loss_log)
```

4.4 Main interfaces

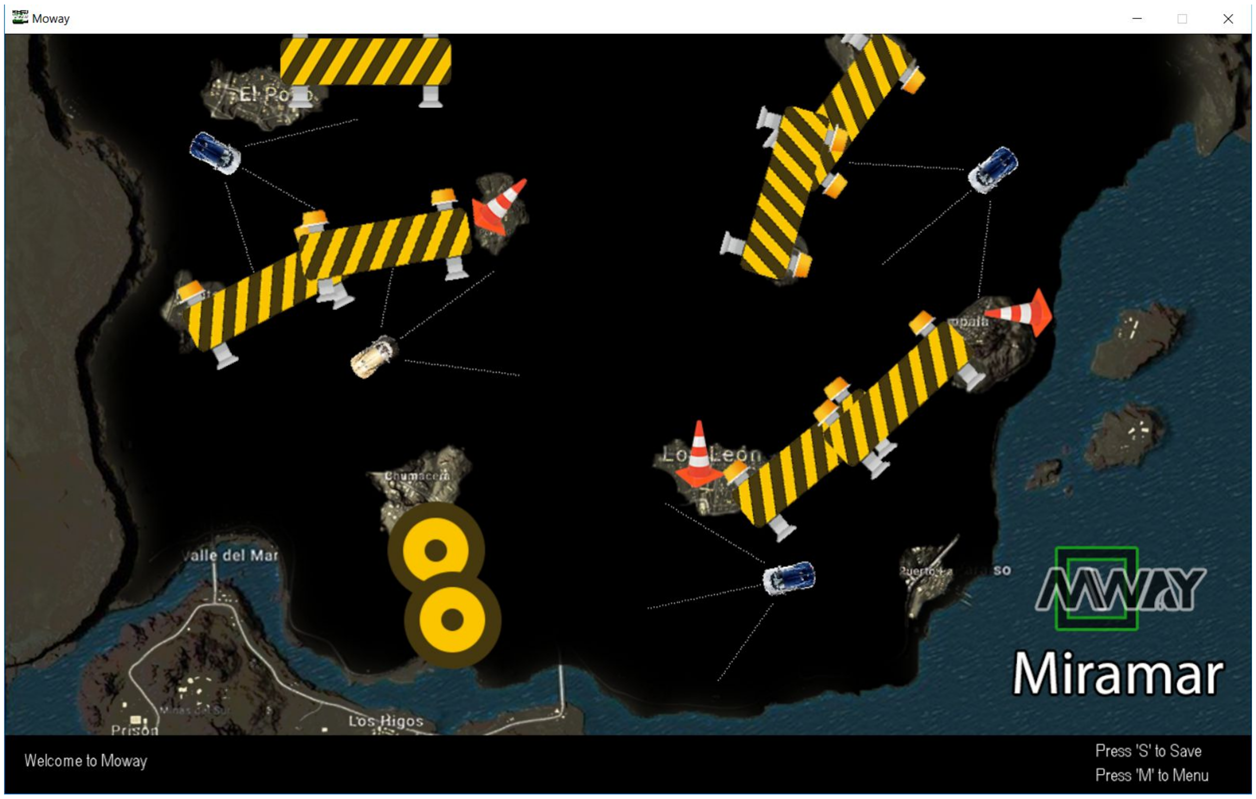




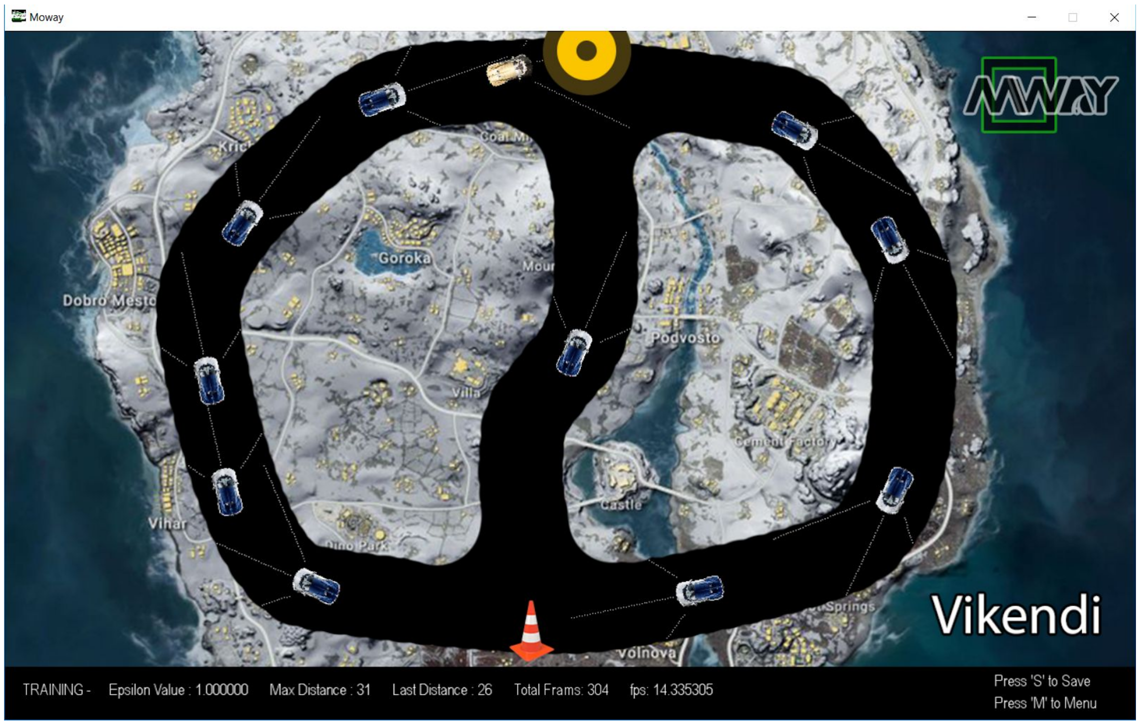
Map : Erangel Object Add : Autonomous-Car Change the Map Using RIGHT, LEFT Arrows Change the Objects Using UP, DOWN Arrows Press 'Z' to Undo Press 'M' to Menu Press 'Enter' To NEXT



Map : Sanhok Object Add : Traffic-Cones-2 Change the Map Using RIGHT, LEFT Arrows Change the Objects Using UP, DOWN Arrows Press 'Z' to Undo Press 'M' to Menu Press 'Enter' To NEXT

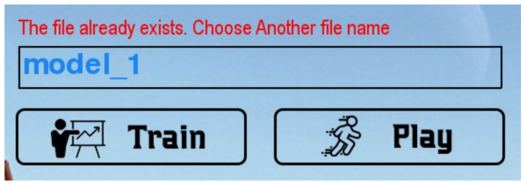


Welcome to Moway Press 'S' to Save Press 'M' to Menu



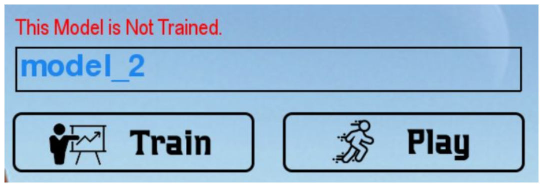
TRAINING - Epsilon Value : 1.000000 Max Distance : 31 Last Distance : 26 Total Frams: 304 fps: 14.335305

Press 'S' to Save
Press 'M' to Menu



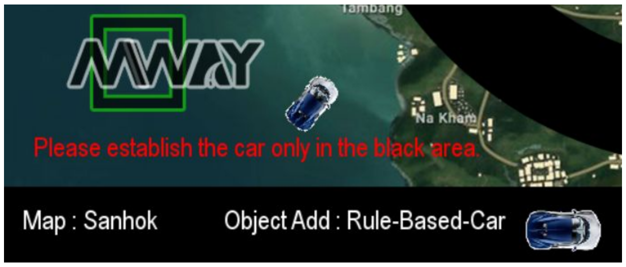
The file already exists. Choose Another file name

model_1



This Model is Not Trained.

model_2



Please establish the car only in the black area.

Map : Sanhok Object Add : Rule-Based-Car

First of all User meet the Menu interface and user wants to enter the Model name that he want to Train or play. Then user meet Environment creation menu interface here first User can Change The map using Right and Left Arrow Keys, And also user can Change the adding obstacle or manned car using Up and Down Arrow Keys. User can set the location of the obstacle by using mouse Click button and also set the angle of obstacle by releasing the mouse button. After user can Train and play the model.

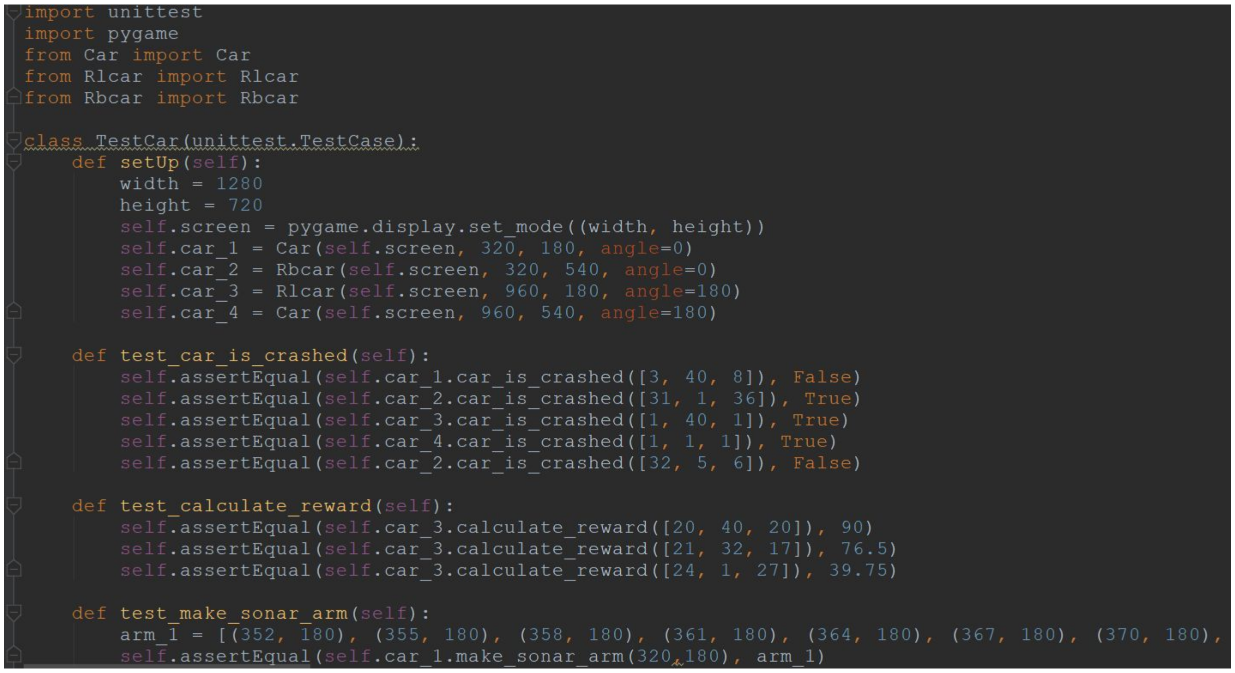
5. System Testing and Analysis

5.1 Testing approach

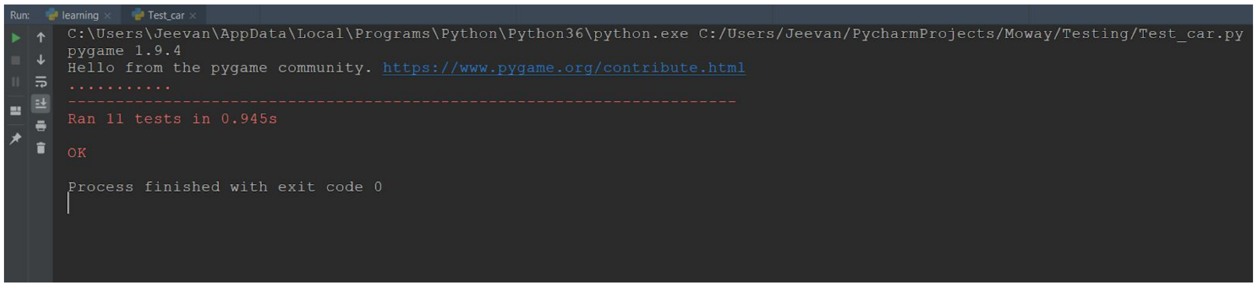
Every system should be thoroughly tested to identify defects and to see whether the system satisfies the customer requirements as intended. In parallel to the implementation of the application, testing was carried out as a major activity. Different types of testing used can be described as below. Moway also was tested prior to deployment. The testing approach was iterative testing in which, once each component is finished it was tested before and after integrating with the whole system. Hence, there was a testing phase at the end of each iteration. Testing was done covering four major aspects of the system; function testing, security testing, performance testing and UI testing. The outcome of each testing category is described below.

5.2 Unit Testing, Results and Analysis of Testing

Unit testing tests individual components to check whether they perform as intended. When doing the function testing, unit testing was involved too. Unit testing was performed for all the data access objects to test their methods. Also, the Car class, which implements the algorithms, was also tested. Unit testing was performed using unittest library and JetBrains PyCharm Community Edition.



```python
import unittest
import pygame
from Car import Car
from Rlcar import Rlcar
from Rbcar import Rbcar

class TestCar(unittest.TestCase):
    def setUp(self):
        width = 1280
        height = 720
        self.screen = pygame.display.set_mode((width, height))
        self.car_1 = Car(self.screen, 320, 180, angle=0)
        self.car_2 = Rbcar(self.screen, 320, 540, angle=0)
        self.car_3 = Rlcar(self.screen, 960, 180, angle=180)
        self.car_4 = Car(self.screen, 960, 540, angle=180)

    def test_car_is_crashed(self):
        self.assertEqual(self.car_1.car_is_crashed([3, 40, 8]), False)
        self.assertEqual(self.car_2.car_is_crashed([31, 1, 36]), True)
        self.assertEqual(self.car_3.car_is_crashed([1, 40, 1]), True)
        self.assertEqual(self.car_4.car_is_crashed([1, 1, 1]), True)
        self.assertEqual(self.car_2.car_is_crashed([32, 5, 6]), False)

    def test_calculate_reward(self):
        self.assertEqual(self.car_3.calculate_reward([20, 40, 20]), 90)
        self.assertEqual(self.car_3.calculate_reward([21, 32, 17]), 76.5)
        self.assertEqual(self.car_3.calculate_reward([24, 1, 27]), 39.75)

    def test_make_sonar_arm(self):
        arm_1 = [(352, 180), (355, 180), (358, 180), (361, 180), (364, 180), (367, 180), (370, 180),
        self.assertEqual(self.car_1.make_sonar_arm(320,180), arm_1)
```



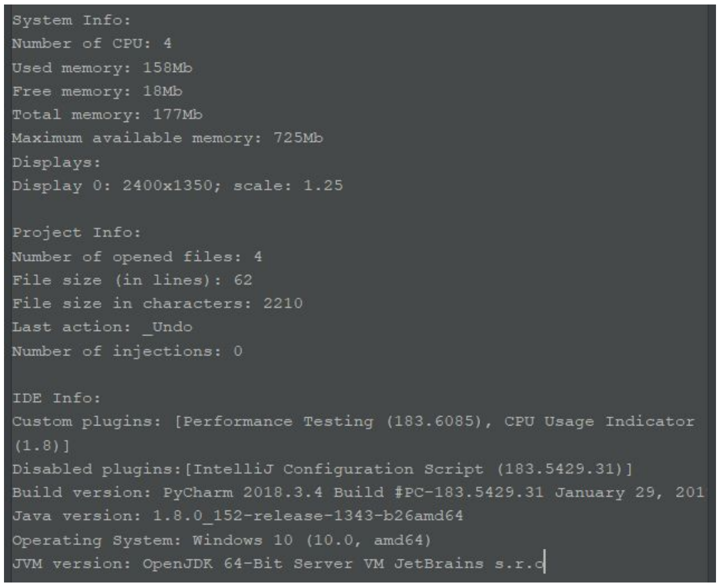
```
C:\Users\Jeevan\AppData\Local\Programs\Python\Python36\python.exe C:/Users/Jeevan/PycharmProjects/Moway/Testing/Test_car.py
pygame 1.9.4
Hello from the pygame community. https://www.pygame.org/contribute.html
...........
----------------------------------------------------------------------
Ran 11 tests in 0.945s

OK

Process finished with exit code 0
```

As shown above, all the test cases were passed thereby consolidating the behavior of all the database access methods. Also, each and every functionality of the system was tested via a technique called “black-box testing” in which the results of executing the use case scenarios are verified by interacting with the Graphical User Interfaces of the system. All functionalities were tested giving both valid and invalid data to see whether the correct feedback is received

5.3 Aspects Related to Performance, User Interface, Security and Failures

Code Metrics Analysis - This is a tool which helps to make the code more maintainable. This detects where code needs rework or increased testing. This includes analyzing maintainability index, Cyclomatic Complexity, Depth of inheritance and Class coupling. Having the maintainability index between 20 and 100 (green) indicates that the code has good maintainability which means the code is easy to maintain. Low coupling and high cohesion are good since they indicate that the design is easy to reuse and maintain because of its less independence on other types

Performance and Diagnostic Analyzer- This is helpful to analyze UI responsiveness which is the ability for the app to be fast and fluid. This also includes analyzing memory usage, energy consumption and CPU usage. This analyzer showed that loading the data file whenever the user selects an option on main page is expensive. As a solution, the code was modified to load the data file only once at the time of launching the main page.



```
System Info:
Number of CPU: 4
Used memory: 158Mb
Free memory: 18Mb
Total memory: 177Mb
Maximum available memory: 725Mb
Displays:
Display 0: 2400x1350; scale: 1.25

Project Info:
Number of opened files: 4
File size (in lines): 62
File size in characters: 2210
Last action: _Undo
Number of injections: 0

IDE Info:
Custom plugins: [Performance Testing (183.6085), CPU Usage Indicator
(1.8)]
Disabled plugins:[IntelliJ Configuration Script (183.5429.31)]
Build version: PyCharm 2018.3.4 Build #PC-183.5429.31 January 29, 201
Java version: 1.8.0_152-release-1343-b26amd64
Operating System: Windows 10 (10.0, amd64)
JVM version: OpenJDK 64-Bit Server VM JetBrains s.r.o
```

No of RB- Car	0	1	2	4	6	8	10	12	14	16
fps	26.42	25.83	24.83	23.48	22.37	20.6 8	17.59	15.44	13.27	11.88

6. Conclusion and Future Work

Detailed description of the project development and tastings are given in this report. During the initial stage of the project development stage I faced many problems in understanding the technology and keeping up the schedule. But the when the time moved on, I was able to easily understand the technologies. According to the schedule given in the introduction, I was able to finish my project as scheduled. Since this is the first project which I have done alone from the start to end, during the development of project I gained a deep understanding of each software engineering principles and why they are so important.

I got the practical experience of importance of various documentations such as software requirements specification and design specification. Doing the project on time has the same importance as making the system to work correctly. I tried to practice to work according to the schedule in order to finish it on time.

Altogether this application development for the software engineering project module gave me a deep understanding of good practices of software engineering. As I have mentioned above I believe that I can take this application to next stage by publishing it in the online Markets.

References

[1] Rmgi.blog. (2019). pygame 2d car tutorial. [online] Available at: http://rmgi.blog/pygame-2d-car-tutorial.html [Accessed 16 Feb. 2019].

[2] Coastline Automation. (2019). Using reinforcement learning in Python to teach a virtual car to avoid obstacles. [online] Available at: https://blog.coast.ai/using-reinforcement-learning-in-python-to-teach-a-virtual-car-to-avoid-obstacles-6e782cc7d4c6 [Accessed 2 Mar. 2019].

[3] PAN, X., YOU, Y., WANG, Z. AND LU, C.
Virtual to Real Reinforcement Learning for Autonomous Driving
In-text: (Pan et al., 2019)

[4] Journal STILGOE, J.
Machine learning, social learning and the governance of self-driving cars
In-text: (Stilgoe, 2017)

[5]WU, C., KREIDIEH, A., PARVATE, K., VINITSKY, E. AND BAYEN, A. M.
Flow: Architecture and Benchmarking for Reinforcement Learning in Traffic Control In-text: (Wu et al., 2019)