CS6611 - Creative and Innovative Project

Project Guide : Dr.S.Muthurajkumar

Team number: 09

Path Navigation using Deep Q Learning

Team Members

Rahul B (2019503545)

Sanjay kumar L S (2019503042)

Gokkul E(2019503517)

Table of Contents :

Abstract :

There is an increased need for automation in this ever advancing technological world. Automation reduces time, money, labour, while also reducing manual errors, giving you more time to concentrate on other work. Manual tasks are hectic and boring, sometimes they are dangerous considering the work. Navigation plays a vital role in many industries, some of them are vehicles, drones, transportation, and there are certain cases where navigation might be dangerous like deep forest, underground, underwater, firefighting environments. Visually challenged people have a difficult time navigating unfamiliar places. So there is a necessity for an automatic navigation system to overcome these situations.Hence in our project we will be implementing automatic path navigation simulation using deep Q-learning (a Machine Learning technique).This path navigation system when assigned its destination automatically finds the shortest route and avoid the obstacle in order to reach its destination

Introduction :

A path navigation system using Deep Q learning is presented in this project. Initially an environment is created with the help of Kivy software Then an object is created with 3 sensors .The object will have to avoid obstacles that are created dynamically by the user and reach its destination by taking the shortest path . Here we will be using Deep Q Learning algorithm (Reinforcement Learning) wherein the object will be trained based on reward and penalty mechanism.The main aim of this project is to make an object automatically navigate to its destination by avoiding the obstacles .

Objective :

- To build an environment with an object and 3 sensors(one right sensor ,one left sensor and one straight sensor.
- To build an obstacle creator and assign a function to the object (i.e.)The object has to turn left or right in order to avoid the obstacle
- To implement deep Q learning algorithm to train the object in order to avoid the obstacle and find the shortest path to reach its destination

Literature Survey :

S.N	***Title of the paper and Published year***	***Proposed Work and Results***	***Limitations***
1	**“Autonomous Vehicle for Obstacle Detection and Avoidance Using Reinforcement**	They have developed a static obstacle detection using reinforcement learning for autonomous vehicle	Although their work tells us about an efficient way for static obstacle detection it still doesn’t work for a

	Learning” **C.S.Arvind, J.Senthilnath**	navigation in a simulated environment.. MLP-NN will be predicting the next action based on vehicle acceleration, heading angle, distance measure from the ultrasonic sensor.	dynamic moving obstacles
2	**“An Autonomous path finding Robot using Q-Learning “** **Madhu Babu V , Vamshi Krishna U , Shahensha S K**	They have implemented a path and motion planning for a robot were used to make it autonomous in unknown environment.These were achieved using image processing and reinforcement techniques using Q Learning .They have calculated the hottest	They have adopted a basic method of edge detection for tracing obstacles on smooth surfaces . However if the surface has any unidentified intensity variations makes it to detect as an obstacle

		path from current state to goal state by analyzing the environment through captured images	
3	**“Q-Learning Algorithms: A Comprehensive Classification and Applications”** **BEAKCHEOL JANG , ,MYEONGHWI KIM ,** **GASPARD HARERIMANA ,** **AND JONG WOOK KIM**	Q-learning algorithms are off policy reinforcement learning algorithms that try to perform the most profitable action given the current state .They covered all variants of Q-learning algorithms, which are a representative algorithm under reinforcement learning. They have distinctively categorized Q-learning algorithms into single-agent and	A major limitation of Q-learning is that it is only works in environments with discrete and finite state and action spaces. Drawbacks or disadvantages of Deep Learning It requires a very large amount of data in order to perform better than other techniques.

		multi-agent and described them thoroughly. Deep Q learning came as an improved version on basic Q learning	
4	**“ Path Planning for Intelligent Robots Based on Deep Q-learning With Experience Replay and Heuristic Knowledge”** **Lan Jiang, Hongyun Huang, and Zuohua Ding**	They have combined deep Q learning with learning replay and heuristic knowledge for path detection and obstacle avoidance of intelligent robots.	The system simulation is hard to satisfy in real life,due to uncertainties in the environment and cannot be applied on an dynamic moving obstacles
5	**“ Decision-Making Strategy on Highway for Autonomous Vehicles Using Deep**	A DRL enabled highway overtaking driving policy is constructed for autonomous vehicles. The proposed	The DQL and DDQN algorithms are compared and analyzed theoretically but practically when

	Reinforcement Learning " **JIANGDONG LIAO , TENG LIU , XIAOLIN TANG , XINGYU MU , BING HUANG , DONGPU CAO**	decision-making strategy is evaluated and estimated to be adaptive to other complicated scenarios.First, the studied driving environment is founded on the highway, wherein an ego vehicle aims to run through a particular driving scenario efficiently and safely. Finally, the performance of the proposed control framework is discussed via executing a series of simulation experiments.	random increase in speed could lead to failure.
6	**Multi-Robot Path Planning Method Using**	They dealt with information and strategy around	The environment where the generated path is

	Reinforcement Learning **Hyansu Bae, Gidong Kim, Jonguk Kim, Dianwei Qian and Sukgyu Lee.**	reinforcement learning for multi-robot navigation algorithms where each robot can be considered as a dynamic obstacle or cooperative robot depending on the situation. That is, each robot in the system can perform independent actions and simultaneously collaborate with each other depending on the given mission. After the selected action, the relationship with the target is evaluated, and rewards or penalty is given to each robot to learn.	simple or without obstacles, an unnecessary movement occurs.and it did not take into account the dynamics of robots and obstacles.

7	**Robot Training and Navigation through the Deep Q-Learning Algorithm** **Madson Rodrigues Lemos; Anne Vitoria Rodrigues de Souza; Renato Souza de Lira; Carlos Alberto Oliveira de Freitas;**	They aimed to present the results of an assessment of adherence to the Deep Q-learning algorithm, applied to a vehicular navigation robot. The robot's job was to transport parts through an environment, for this purpose, a decision system was built based on the Deep Q-learning algorithm, with the aid of an artificial neural network that received data from the sensors as input and allowed autonomous navigation in an environment. For the experiments, the mobile robot-maintained	The research was limited to the use of educational robots. The algorithm does not perform more complex tests with dynamic environments.

		communication via the network with other robotic components present in the environment through the MQTT protocol.	
8	**Towards Real-Time Path Planning through Deep Reinforcement Learning for a UAV in Dynamic Environments** **Chao Yan, Xiaojia Xiang & Chang Wang**	They have proposed a Deep Reinforcement Learning (DRL) approach for UAV path planning based on the global situation information. They have chosen the STAGE Scenario software to provide the simulation environment where a situation assessment model is developed with consideration of the UAV survival	Although their research is highly efficient in simulation , it is hard to develop this in real life environment and it is not feasible

		probability under enemy radar detection and missile attack.	
9	**Path planning of mobile robot in unknown dynamic continuous environment using reward-modified deep Q-network** **Runnan Huang Chengxuan Qin Jian Ling Li Xuejing Lan**	Their research aimed at the path planning of mobile robots in UDE, a continuous dynamic simulation environment is built in this article. Based on DQN, a reward function with reward weight is designed, and the influence of reward weight has been analysed experimentally. Moreover, the abnormal rewards caused by the relative	Their work focused on the performance of DQN on the policy of the robot's moving direction, hence the velocity and the acceleration of the robot are not considered Moreover, this article does not consider the specific dimension.

		motion between obstacles and robot have been analysed and solved by adding a reward modifier to DQN. The comparative experiment among RMDQN, RMDDQN, dueling RMDQN, and dueling RMDDQN was done, and turns out that the result of RMDDQN is the best.	
10	**Autonomous Navigation for Omnidirectional Robot Based on Deep Reinforcement Learning** **Van Nguyen Thi**	They aimed to illustrate how the Omni robot performs navigation using model-free deep Q learning to navigate in unpredicted environments. It will also explain how to	Their system attempted to find the best route by moving around or near the obstacle several times which is not practical in real life scenarios.

	Thanh , Tien Ngo Manh, Cuong Nguyen Manh, Dung Pham Tien, Manh Tran Van , Duyen Ha Thi Kim and Duy Nguyen Duc	obtain the policy when such a model is unknown in advance by using a virtual environment to conduct in simulation.	

Architecture Diagram :

This architecture diagram clearly depicts our goal and outline structure of our project.

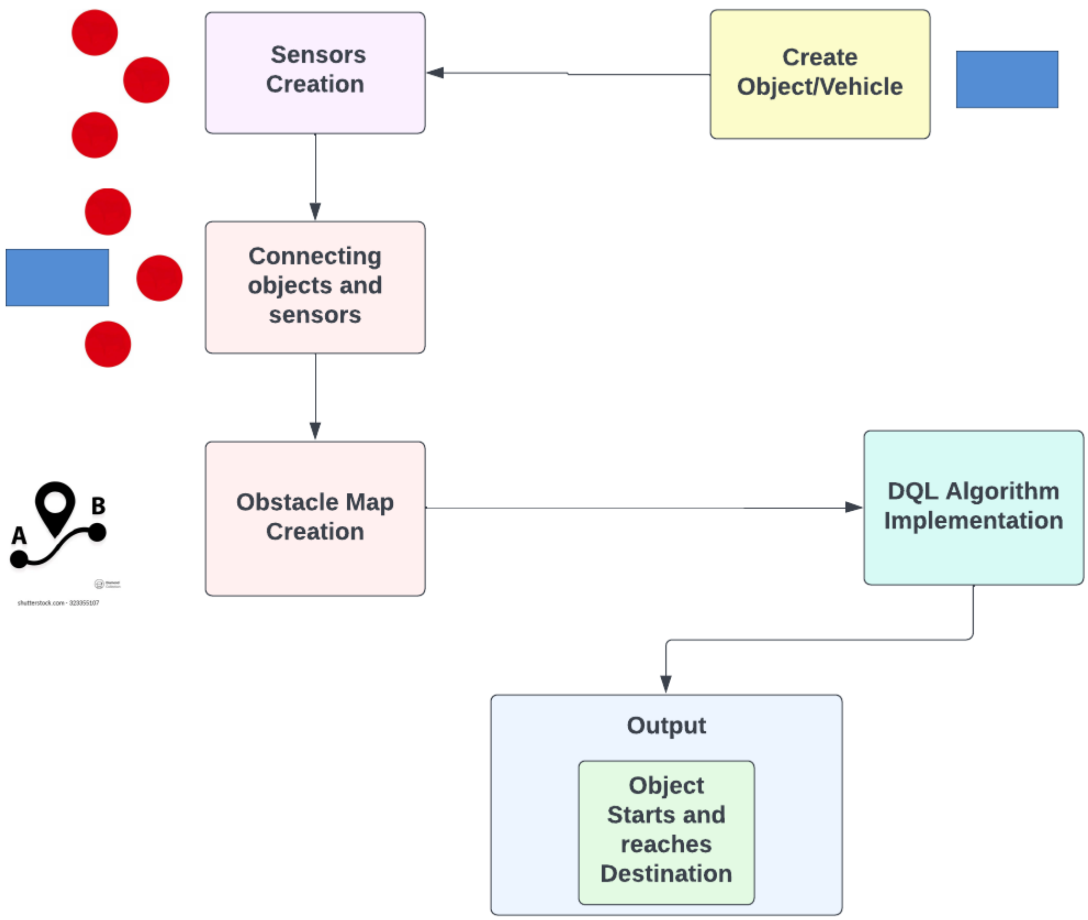

Figure 1 - Architecture Diagram

The project's Architecture diagram in figure 1 has 6 Components ,

1) Create Object/Vehicle :

This module creates the object required, that is the vehicle using Kivy with any desired shape and size .

2) Sensors Creation :

This module creates the 3 sensors ,the left sensor,the right sensor and the straight .The main purpose of these sensors are to detect the obstacles ,if there are any.

3) Connecting objects and sensors :

This module connects the 4 objects into a single entity, that is it connects the object with 3 sensors .

4) Obstacle Map Creation :

In this module, we create an environment/map for the object to navigate from source to destination .We have also added clear(deletes the obstacle) , save (saves the obstacle design) and load (loads the previously saved obstacle design) options .

5) DQL Algorithm Implementation :

This module deals with the DQL Algorithm which is used by the object to learn to avoid obstacles.If the object detects any obstacle from any of the three sensors it turns to the opposite direction by 20 degrees and moves towards its destination. Here reinforcement learning method is used along with deep Q learning where they will be awarded with a reward for every step the move closer to the destination and a penalty will be given when it moves away from the destination and aim of it is to maximise its reward points.

6) Object starts and reaches destination :

This module deals with the object moving from its initial point by dodging all the obstacle and by finding the shortest route to reach to the designated desitination.

Proposed work :

The goal of this simulation is to build an environment with a complex path, an object with 3 sensors which uses Deep Q-Learning to train the object and to assign a destination for the object to reach.

Anaconda is what we'll use to install PyTorch and Kivy. It is a free and open source distribution of Python which offers an easy way to install packages.

We will build this 2D map inside a Kivy webapp. Kivy is a free and open source Python framework with a user interface inside which you can build your games or apps. It will be the container of the whole environment.

PyTorch is the AI framework used to build our Deep Q-Network. PyTorch is great to work with and powerful. It has dynamic graphs which allow fast computations of the gradient of complex functions, needed to train the model.

This can be divided into 3 integral parts to simplify the process ,
1.To create an environment with the object and sensors.
2.To build the obstacle creator and to assign functions for the objects.
3.Implementing Deep Q-Learning to train the object.

Modules :

Module 1 : To create an environment with the object and sensors.

We create the environment and we use Kivy WebApp to create 4 Kivy objects, a rectangle shape representing the object and three sensors to detect any obstacle and to navigate to the destination.

We set our object to go from the upper left corner of the map, to the bottom right corner.

Create 3 buttons: Clear, Load and Save.

Module 2 : To build the obstacle creator and to assign functions for the objects.

We build a system to draw different obstacles in the environment.

We assign functions to the objects to make it go through any path we create from the start to the end point.

Assign function to Clear button.

Module 3 : Implementing Deep Q-Learning to train the object.

Using Deep Q-Learning we build and train our object to navigate its way avoiding any obstacles to its destination.

Assign functions for Load and Save buttons.

Algorithm :

Step 1 : Importing the libraries and the Kivy packages.

Step 2 : Initialising variables to keep the last point in memory when we draw the sand on the map , the total number of points in the last drawing , the length of the last drawing.

Step 3 : Create the brain of our AI, list of actions and the reward variable :

I. 4 inputs, 3 actions, gamma = 0.9.
II. action = 0 => no rotation, action = 1 => rotate 20 degrees, action = 2 => rotate -20 degrees.
III. The reward received after reaching a new state.

Step 4 : Initialising the map :

I. Sand is an array that has as many cells as our graphic interface has pixels. Each cell has a one if there is sand, 0 otherwise.
II. Building x-coordinate and y-coordinate of the goal.
III. Initializing the sand array with only zeros.
IV. The goal to reach is at the upper left of the map. (the x-coordinate and y-coordinate)
V. Initializing the last distance from the object to the goal.

Step 5 : Creating the object class :

I. Initializing the angle of the object.
II. Initializing the last rotation of the object.
III. Initializing the x-coordinate and y-coordinate of the velocity vector and the velocity vector.

IV. Initializing the x-coordinate and y-coordinate of all 3 sensors and their respective sensor vectors.

Step 6 : Updating the position of the object according to its last position and velocity :

I. Getting the rotation of the object.
II. Updating the angle and the position of sensors 1,2 and 3.
III. Updating the signal received by sensors 1,2 and 3. (density of sand around sensor 1,2,3)
IV. If any sensor is out of the map (the object is facing one edge of the map) that sensor detects full sand.
V. Update sensors 1,2 and 3.

Step 7 : Creating the game class :

I. Getting the object and the sensors 1,2 and 3 from our kivy file.
II. Starting the object when we launch the application.
III. The object will start at the center of the map.
IV. The object will start to go horizontally to the right with a speed of 6.

Step 8 : Update function that updates everything that needs to be updated at each discrete time when reaching a new state (getting new signals from the sensors) :

I. Specifying the global variables.
II. Store width and height of the map (horizontal edge and vertical edge).
III. Storing the difference of x-coordinates and of y-coordinates between the goal and the object.
IV. Initializing the direction of the object with respect to the goal (if the object is heading perfectly towards the goal, then orientation = 0)

V. Initializing our input state vector, composed of the orientation plus the three signals received by the three sensors.
VI. Updating the weights of the neural network in our ai and playing a new action
VII. Converting the action played (0, 1 or 2) into the rotation angle (0°, 20° or -20°)
VIII. Moving the object according to this last rotation angle
IX. Getting the new distance between the object and the goal right after the object moved
X. Updating the positions of the 3 sensors 1,2 and 3 right after the object moved.

Step 9 : Assigning reward system :

I. If the object is on the sand, it is slowed down (speed = 1) and reward = -1.
II. Otherwise it gets a bad reward of -0.2.
III. However if it is getting closer to the goal it still gets a slightly positive reward of 0.1.
IV. If the object is on any edge of the frame (top,right,bottom,left), it comes back 10 pixels away from the edge and it gets a bad reward of -1.
V. When the object reaches its goal, the goal becomes the bottom right corner of the map and vice versa (updating of the x and y coordinate of the goal).
VI. Updating the last distance from the object to the goal.

Step 10 : Painting for graphic interface :

I. Putting some sand when we do a left click.
II. Put some sand when we move the mouse while pressing left.

Step 11 : API and switches interface :

I. Building the app.
II. Creating the clear, save and load buttons.
III. Running the app.

Step 12 : Build and run the application.

Project Code :

1) File Name - car.kv (Kivy File) :

```
<Car>:
    size: 20, 10
    canvas:
        PushMatrix
        Rotate:
            angle: self.angle
            origin: self.center
        Rectangle:
            pos: self.pos
            size: self.size
        PopMatrix

<Ball1>:
    size: 10,10
    canvas:
        Color:
            rgba: 1,0,0,1
        Ellipse:
            pos: self.pos
            size: self.size
<Ball2>:
    size: 10,10
```

```
    canvas:
        Color:
            rgba: 0,1,1,1
        Ellipse:
            pos: self.pos
            size: self.size

<Ball3>:
    size: 10,10
    canvas:
        Color:
            rgba: 1,1,0,1
        Ellipse:
            pos: self.pos
            size: self.size

<Game>:
    car: game_car
    ball1: game_ball1
    ball2: game_ball2
    ball3: game_ball3

    Car:
        id: game_car
        center: self.parent.center
    Ball1:
        id: game_ball1
        center: self.parent.center
    Ball2:
        id: game_ball2
        center: self.parent.center
    Ball3:
        id: game_ball3
        center: self.parent.center
```

2) File Name - ai.py (Python File) :

```
# AI for Path Navigation

# Importing the libraries

import numpy as np
import random
import os
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.autograd as autograd
from torch.autograd import Variable

# Creating the architecture of the Neural Network

class Network(nn.Module):

    def __init__(self, input_size, nb_action):
        super(Network, self).__init__()
        self.input_size = input_size
        self.nb_action = nb_action
        self.fc1 = nn.Linear(input_size, 30)
        self.fc2 = nn.Linear(30, nb_action)

    def forward(self, state):
        x = F.relu(self.fc1(state))
        q_values = self.fc2(x)
        return q_values
```

```
# Implementing Experience Replay

class ReplayMemory(object):

    def __init__(self, capacity):
        self.capacity = capacity
        self.memory = []

    def push(self, event):
        self.memory.append(event)
        if len(self.memory) > self.capacity:
            del self.memory[0]

    def sample(self, batch_size):
        samples = zip(*random.sample(self.memory, batch_size))
        return map(lambda x: Variable(torch.cat(x, 0)), samples)

# Implementing Deep Q Learning

class Dqn():

    def __init__(self, input_size, nb_action, gamma):
        self.gamma = gamma
        self.reward_window = []
        self.model = Network(input_size, nb_action)
        self.memory = ReplayMemory(100000)
        self.optimizer = optim.Adam(self.model.parameters(), lr = 0.001)
        self.last_state = torch.Tensor(input_size).unsqueeze(0)
        self.last_action = 0
        self.last_reward = 0

    def select_action(self, state):
            probs = F.softmax(self.model(Variable(state, volatile = True))*100) #
T=100
```

```
        action = probs.multinomial(num_samples=1)
        return action.data[0,0]

    def learn(self, batch_state, batch_next_state, batch_reward, batch_action):
        outputs = self.model(batch_state).gather(1, batch_action.unsqueeze(1)).squeeze(1)
        next_outputs = self.model(batch_next_state).detach().max(1)[0]
        target = self.gamma*next_outputs + batch_reward
        td_loss = F.smooth_l1_loss(outputs, target)
        self.optimizer.zero_grad()
        td_loss.backward(retain_graph = True)
        self.optimizer.step()

    def update(self, reward, new_signal):
        new_state = torch.Tensor(new_signal).float().unsqueeze(0)
        self.memory.push((self.last_state, new_state, torch.LongTensor([int(self.last_action)]), torch.Tensor([self.last_reward])))
        action = self.select_action(new_state)
        if len(self.memory.memory) > 100:
            batch_state, batch_next_state, batch_action, batch_reward = self.memory.sample(100)
            self.learn(batch_state, batch_next_state, batch_reward, batch_action)
        self.last_action = action
        self.last_state = new_state
        self.last_reward = reward
        self.reward_window.append(reward)
        if len(self.reward_window) > 1000:
            del self.reward_window[0]
        return action

    def score(self):
        return sum(self.reward_window)/(len(self.reward_window)+1.)
```

```python
    def save(self):
        torch.save({'state_dict': self.model.state_dict(),
                    'optimizer' : self.optimizer.state_dict(),
                   }, 'last_brain.pth')

    def load(self):
        if os.path.isfile('last_brain.pth'):
            print("=> loading checkpoint... ")
            checkpoint = torch.load('last_brain.pth')
            self.model.load_state_dict(checkpoint['state_dict'])
            self.optimizer.load_state_dict(checkpoint['optimizer'])
            print("done !")
        else:
            print("no checkpoint found...")
```

3) File Name - map.py (Python File) :

```python
# Path Navigation

# Importing the libraries
import numpy as np
from random import random, randint
import matplotlib.pyplot as plt
import time

# Importing the Kivy packages
from kivy.app import App
from kivy.uix.widget import Widget
from kivy.uix.button import Button
from kivy.graphics import Color, Ellipse, Line
from kivy.config import Config
```

```
from kivy.properties import NumericProperty, ReferenceListProperty,
ObjectProperty
from kivy.vector import Vector
from kivy.clock import Clock

# Importing the Dqn object from our AI in ai.py
from ai import Dqn

# Adding this line if we don't want the right click to put a red point
Config.set('input', 'mouse', 'mouse,multitouch_on_demand')

# Introducing last_x and last_y, used to keep the last point in memory when
we draw the sand on the map
last_x = 0
last_y = 0
n_points = 0
length = 0

# Getting our AI, which we call "brain", and that contains our neural network
that represents our Q-function
brain = Dqn(5,3,0.9)
action2rotation = [0,20,-20]
last_reward = 0
scores = []

# Initializing the map
first_update = True
def init():
    global sand
    global goal_x
    global goal_y
    global first_update
    sand = np.zeros((longueur,largeur))
    goal_x = 20
    goal_y = largeur - 20
```

```
    first_update = False

# Initializing the last distance
last_distance = 0

# Creating the car class

class Car(Widget):

    angle = NumericProperty(0)
    rotation = NumericProperty(0)
    velocity_x = NumericProperty(0)
    velocity_y = NumericProperty(0)
    velocity = ReferenceListProperty(velocity_x, velocity_y)
    sensor1_x = NumericProperty(0)
    sensor1_y = NumericProperty(0)
    sensor1 = ReferenceListProperty(sensor1_x, sensor1_y)
    sensor2_x = NumericProperty(0)
    sensor2_y = NumericProperty(0)
    sensor2 = ReferenceListProperty(sensor2_x, sensor2_y)
    sensor3_x = NumericProperty(0)
    sensor3_y = NumericProperty(0)
    sensor3 = ReferenceListProperty(sensor3_x, sensor3_y)
    signal1 = NumericProperty(0)
    signal2 = NumericProperty(0)
    signal3 = NumericProperty(0)

    def move(self, rotation):
        self.pos = Vector(*self.velocity) + self.pos
        self.rotation = rotation
        self.angle = self.angle + self.rotation
        self.sensor1 = Vector(30, 0).rotate(self.angle) + self.pos
        self.sensor2 = Vector(30, 0).rotate((self.angle+30)%360) + self.pos
        self.sensor3 = Vector(30, 0).rotate((self.angle-30)%360) + self.pos
```

```
                    self.signal1 = int(np.sum(sand[int(self.sensor1_x)-10:int(self.sensor1_x)+10, int(self.sensor1_y)-10:int(self.sensor1_y)+10]))/400.
                    self.signal2 = int(np.sum(sand[int(self.sensor2_x)-10:int(self.sensor2_x)+10, int(self.sensor2_y)-10:int(self.sensor2_y)+10]))/400.
                    self.signal3 = int(np.sum(sand[int(self.sensor3_x)-10:int(self.sensor3_x)+10, int(self.sensor3_y)-10:int(self.sensor3_y)+10]))/400.
        if self.sensor1_x>longueur-10 or self.sensor1_x<10 or self.sensor1_y>largeur-10 or self.sensor1_y<10:
          self.signal1 = 1.
        if self.sensor2_x>longueur-10 or self.sensor2_x<10 or self.sensor2_y>largeur-10 or self.sensor2_y<10:
          self.signal2 = 1.
        if self.sensor3_x>longueur-10 or self.sensor3_x<10 or self.sensor3_y>largeur-10 or self.sensor3_y<10:
          self.signal3 = 1.

class Ball1(Widget):
   pass
class Ball2(Widget):
   pass
class Ball3(Widget):
   pass

# Creating the game class

class Game(Widget):

   car = ObjectProperty(None)
   ball1 = ObjectProperty(None)
   ball2 = ObjectProperty(None)
   ball3 = ObjectProperty(None)
```

```
    def serve_car(self):
        self.car.center = self.center
        self.car.velocity = Vector(6, 0)

    def update(self, dt):

        global brain
        global last_reward
        global scores
        global last_distance
        global goal_x
        global goal_y
        global longueur
        global largeur

        longueur = self.width
        largeur = self.height
        if first_update:
            init()

        xx = goal_x - self.car.x
        yy = goal_y - self.car.y
        orientation = Vector(*self.car.velocity).angle((xx,yy))/180.
              last_signal  =  [self.car.signal1,  self.car.signal2,  self.car.signal3,
orientation, -orientation]
        action = brain.update(last_reward, last_signal)
        scores.append(brain.score())
        rotation = action2rotation[action]
        self.car.move(rotation)
        distance = np.sqrt((self.car.x - goal_x)**2 + (self.car.y - goal_y)**2)
        self.ball1.pos = self.car.sensor1
        self.ball2.pos = self.car.sensor2
        self.ball3.pos = self.car.sensor3

        if sand[int(self.car.x),int(self.car.y)] > 0:
```

```
            self.car.velocity = Vector(1, 0).rotate(self.car.angle)
            last_reward = -1
        else: # otherwise
            self.car.velocity = Vector(6, 0).rotate(self.car.angle)
            last_reward = -0.2
            if distance < last_distance:
                last_reward = 0.1

        if self.car.x < 10:
            self.car.x = 10
            last_reward = -1
        if self.car.x > self.width - 10:
            self.car.x = self.width - 10
            last_reward = -1
        if self.car.y < 10:
            self.car.y = 10
            last_reward = -1
        if self.car.y > self.height - 10:
            self.car.y = self.height - 10
            last_reward = -1

        if distance < 100:
            goal_x = self.width-goal_x
            goal_y = self.height-goal_y
        last_distance = distance

# Adding the painting tools

class MyPaintWidget(Widget):

    def on_touch_down(self, touch):
        global length, n_points, last_x, last_y
        with self.canvas:
            Color(0.8,0.7,0)
            d = 10.
```

```
            touch.ud['line'] = Line(points = (touch.x, touch.y), width = 10)
            last_x = int(touch.x)
            last_y = int(touch.y)
            n_points = 0
            length = 0
            sand[int(touch.x),int(touch.y)] = 1

    def on_touch_move(self, touch):
        global length, n_points, last_x, last_y
        if touch.button == 'left':
            touch.ud['line'].points += [touch.x, touch.y]
            x = int(touch.x)
            y = int(touch.y)
            length += np.sqrt(max((x - last_x)**2 + (y - last_y)**2, 2))
            n_points += 1.
            density = n_points/(length)
            touch.ud['line'].width = int(20 * density + 1)
                sand[int(touch.x) - 10 : int(touch.x) + 10, int(touch.y) - 10 :
int(touch.y) + 10] = 1
            last_x = x
            last_y = y

# Adding the API Buttons (clear, save and load)

class CarApp(App):

    def build(self):
        parent = Game()
        parent.serve_car()
        Clock.schedule_interval(parent.update, 1.0/60.0)
        self.painter = MyPaintWidget()
        clearbtn = Button(text = 'clear')
        savebtn = Button(text = 'save', pos = (parent.width, 0))
        loadbtn = Button(text = 'load', pos = (2 * parent.width, 0))
        clearbtn.bind(on_release = self.clear_canvas)
```

```
        savebtn.bind(on_release = self.save)
        loadbtn.bind(on_release = self.load)
        parent.add_widget(self.painter)
        parent.add_widget(clearbtn)
        parent.add_widget(savebtn)
        parent.add_widget(loadbtn)
        return parent

    def clear_canvas(self, obj):
        global sand
        self.painter.canvas.clear()
        sand = np.zeros((longueur,largeur))

    def save(self, obj):
        print("saving brain...")
        brain.save()
        plt.plot(scores)
        plt.show()

    def load(self, obj):
        print("loading last saved brain...")
        brain.load()

# Running the whole thing
if __name__ == '__main__':
    CarApp().run()
```

Tools Used :

- **Kivy** - To create the environment and the objects.

Kivy is an open source multi-platform GUI development library for Python and can run on iOS, Android, Windows, OS X, and GNU/Linux. It helps develop applications that make use of innovative, multi-touch UI. The fundamental idea behind Kivy is to enable the developer to build an app once and use it across all devices, making the code reusable and deployable, allowing for quick and easy interaction design and rapid prototyping.

- **PyTorch Framework** - To implement the Deep Q Learning algorithm and to build the Path Navigating System.

PyTorch is an open source machine learning (ML) framework based on the Python programming language and the Torch library. It is one of the preferred platforms for deep learning research. The framework is built to speed up the process between research prototyping and deployment.

PyTorch is similar to NumPy and computes using tensors that are accelerated by graphics processing units (GPU). Tensors are arrays, a type of multidimensional data structure, that can be operated on and manipulated with APIs. The PyTorch framework supports over 200 different mathematical operations.

The popularity of PyTorch continues to rise as it simplifies the creation of artificial neural network (ANN) models. PyTorch is mainly used for applications of research, data science and artificial intelligence (AI).

Implementation :

Module 1 : To create an environment with the object and sensors.

The environment with objects are created, the object is set to go from the upper left corner of the map to the bottom right corner.

Figure 2 - Random Movement 1

The object starting at the top left of the map(starting point) is shown in Figure 2.

Three buttons are created : Clear, Load and Save.

Figure 3 - Random Movement 2

The object reaching the bottom right of the map(destination) is shown in Figure 3.

Module 2 : To build the obstacle creator and to assign functions for the objects.

Obstacle creator using mouse pointer is built, It allows us to draw different obstacles in the environment on which the object is tested.

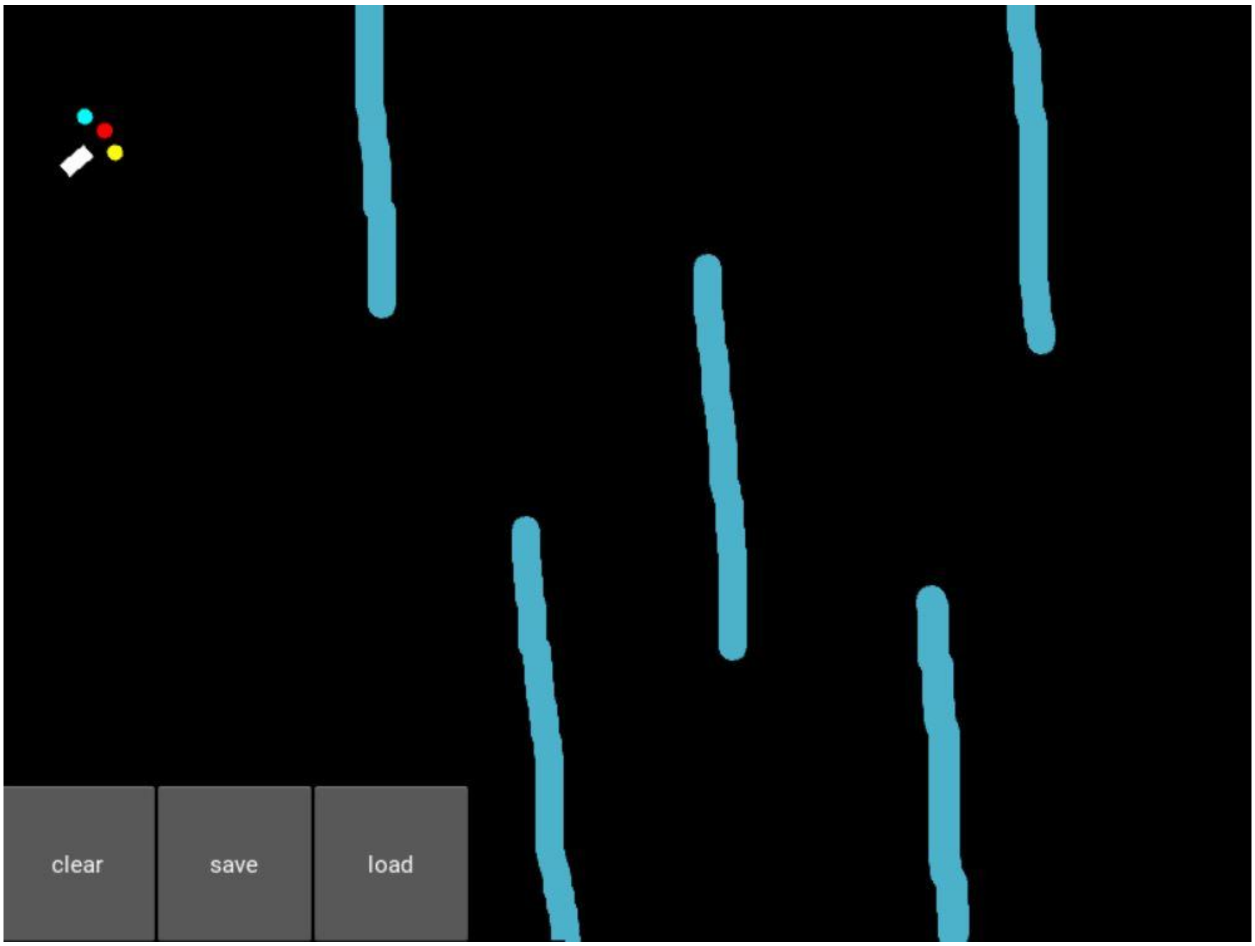

Figure 4 - Obstacle 1 (Object at initial position)

Obstacle is created using the obstacle creator we implemented, it can be hand drawn by us to build various different obstacles as shown in Figure 4.

Functions are assigned to the objects to make it go through any path we create from the start to the end point. The clear can refresh the obstacle.

Figure 5 - Obstacle 1(Object traveling through obstacle)

The movement of the object through the obstacle is shown in Figure 5.

Module 3 : Implementing Deep Q-Learning to train the object.

We built and trained our object to navigate its way avoiding any obstacles to its destination using Deep Q-Learning.

Assign functions for Load and Save buttons.

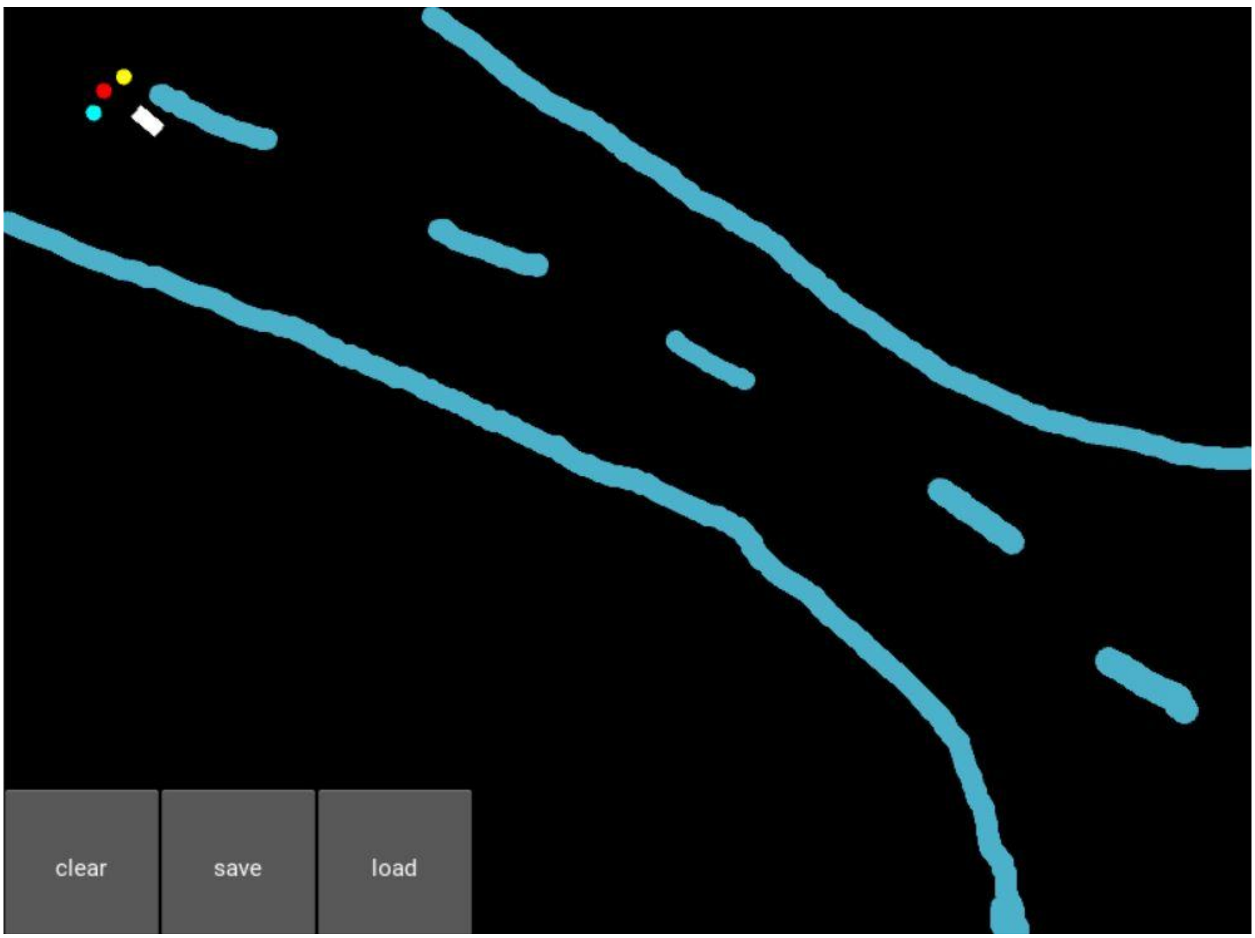

Figure 6 - Obstacle 2 (Object at the initial position)

Figure 7- Obstacle 2(Object travelling)

Figure 7 shows the object travelling through the obstacle , to reach the destination.

Figure 8 - Obstacle 2(Object reaching the destination)

Deep Q-Learning is used to train the object to learn the best route for the object to reach the destination, it checks all possible routes and then follows the best route , this is shown in figures 6, 7 and 8.

Functions are assigned for the load and save buttons.

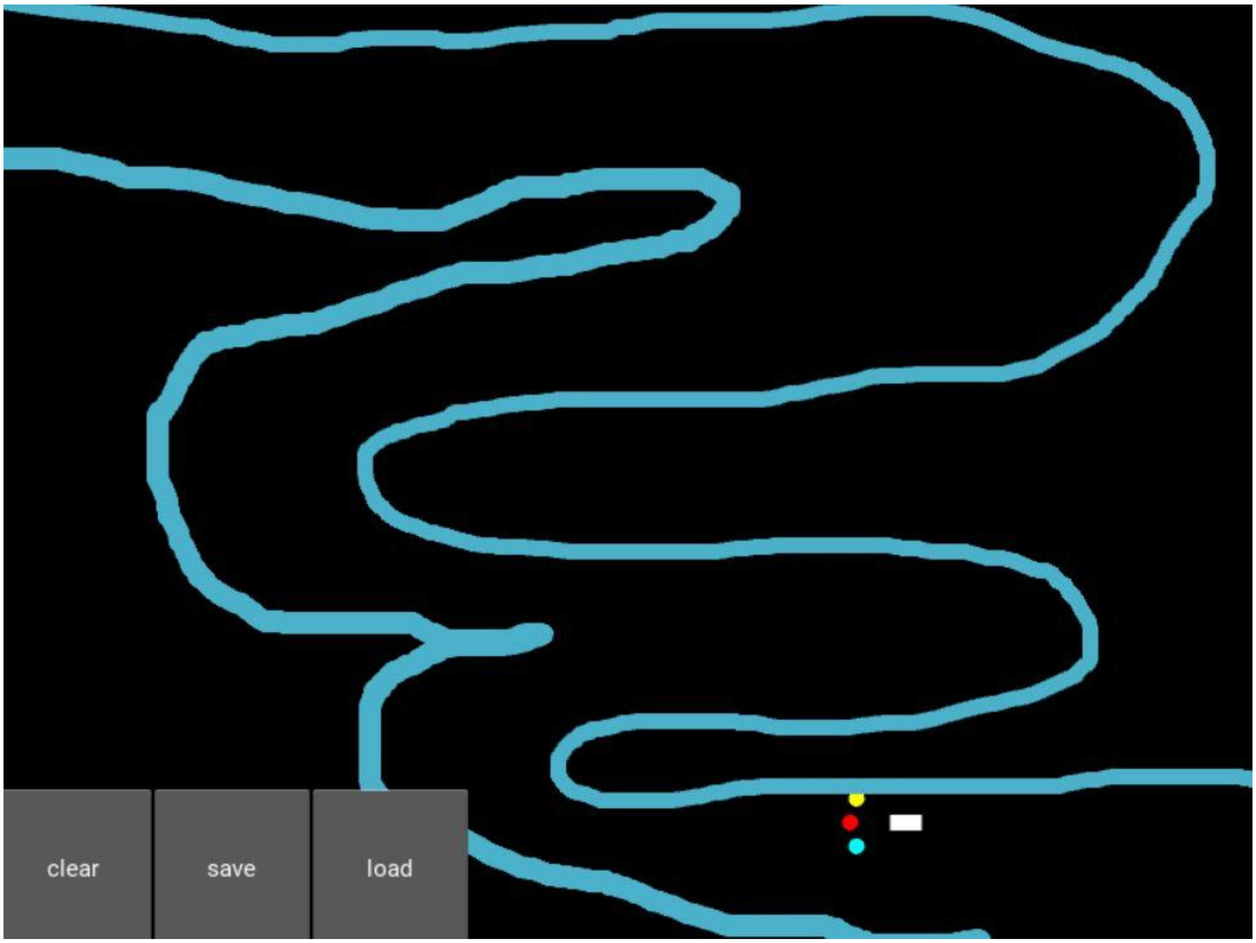

Figure 9 - Obstacle 3(Object starting at destination)

Figure 9 shows an object returning from the destination after reaching it, the program changes or alternates the start and destination points after a travel from start to end is completed.

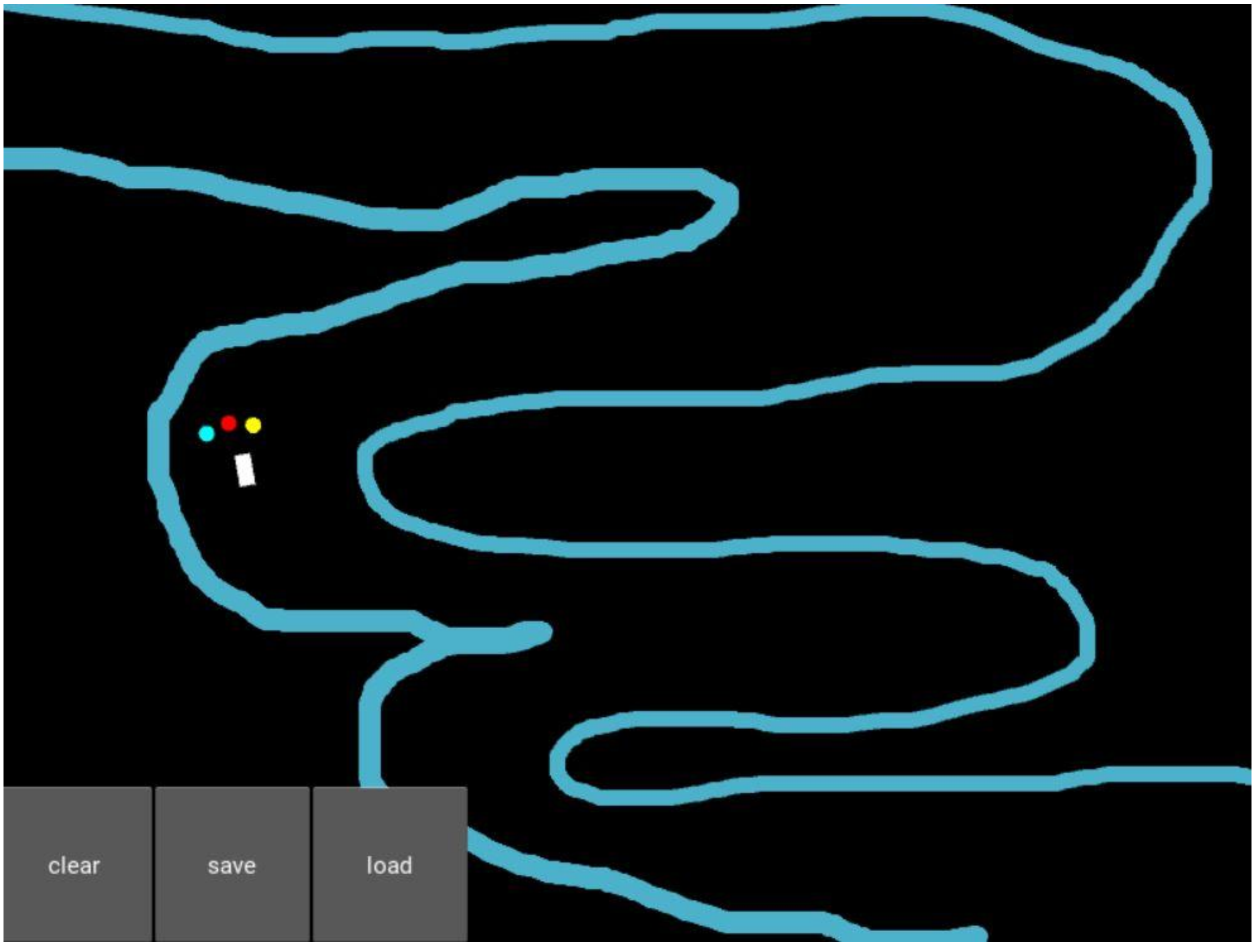

Figure 10 - Obstacle 3(Object Traversing)

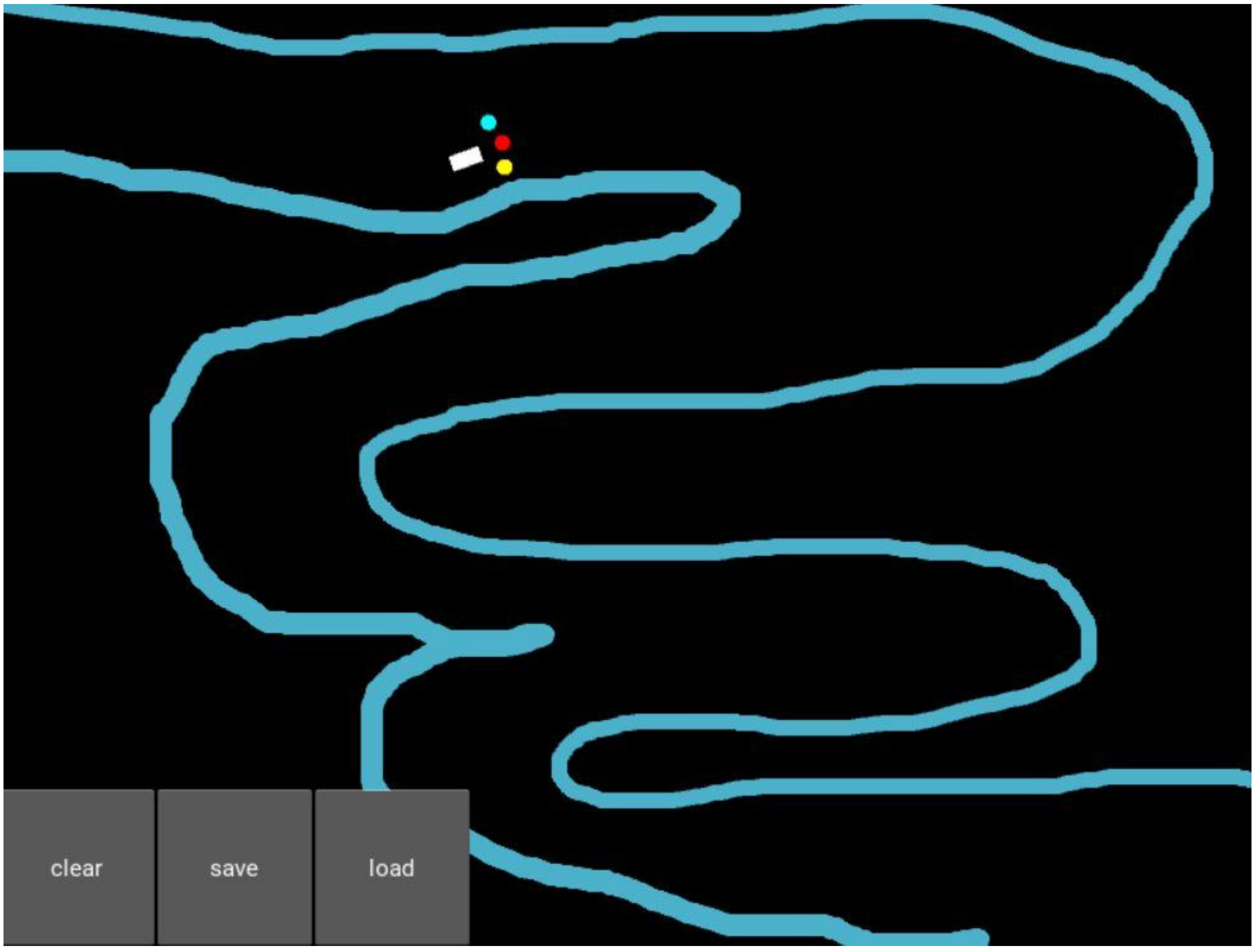

Figure 11 - Obstacle 3(Object reaching its destination)

Figures 10 and 11 shows the object learning the environment, and it reaches the starting point back from the destination. This process continues repeatedly as the object learns the environment better and better. Finally the object follows the best path through the obstacle.

Performance Analysis :

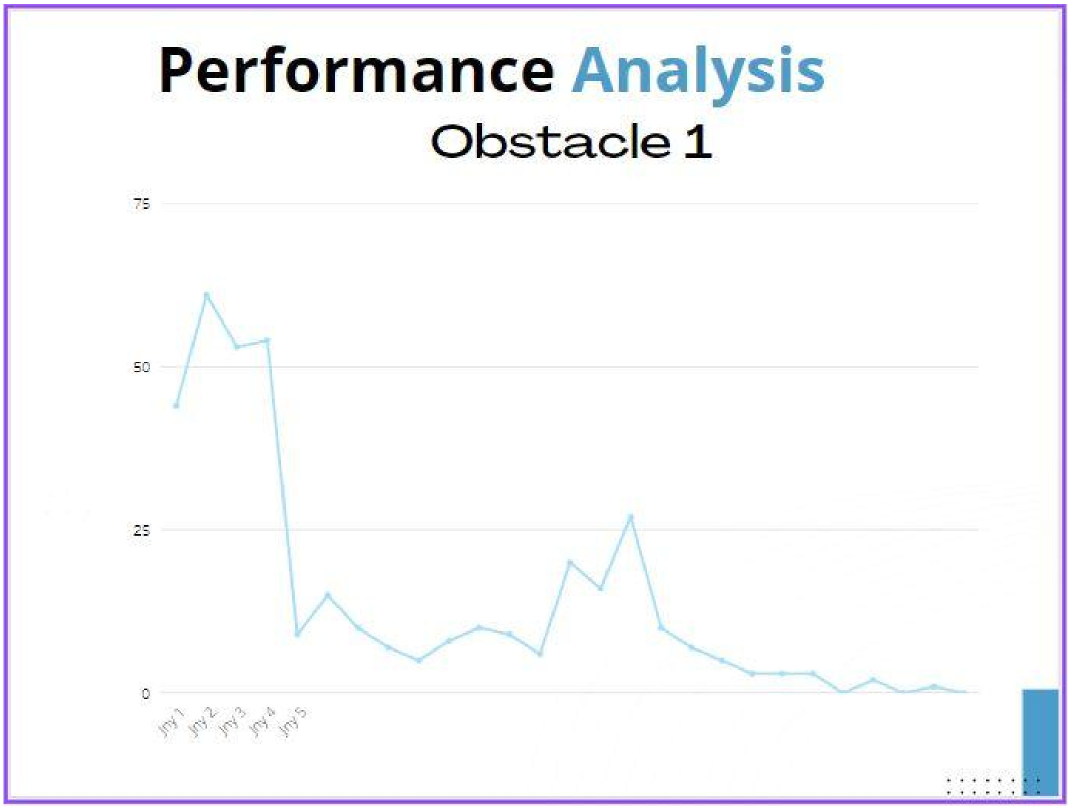

Figure 12 - Performance Analysis for Obstacle 1

Performance analysis is done for the environment in Obstacle-1(fig.5). First we can notice a lot of hits at the obstacle by the object . Once it learnt the environment , we can notice a drastic change in the no.of hits per journey.

Figure 11 - Obstacle 3(Object reaching its destination)

Figure 13 - Performance Analysis for Obstacle 2

Performance analysis is done for the environment in Obstacle-2(fig.6). First we can notice a lot of hits at the obstacle by the object . Once it learnt the environment , we can notice a drastic change in the no.of hits per journey.

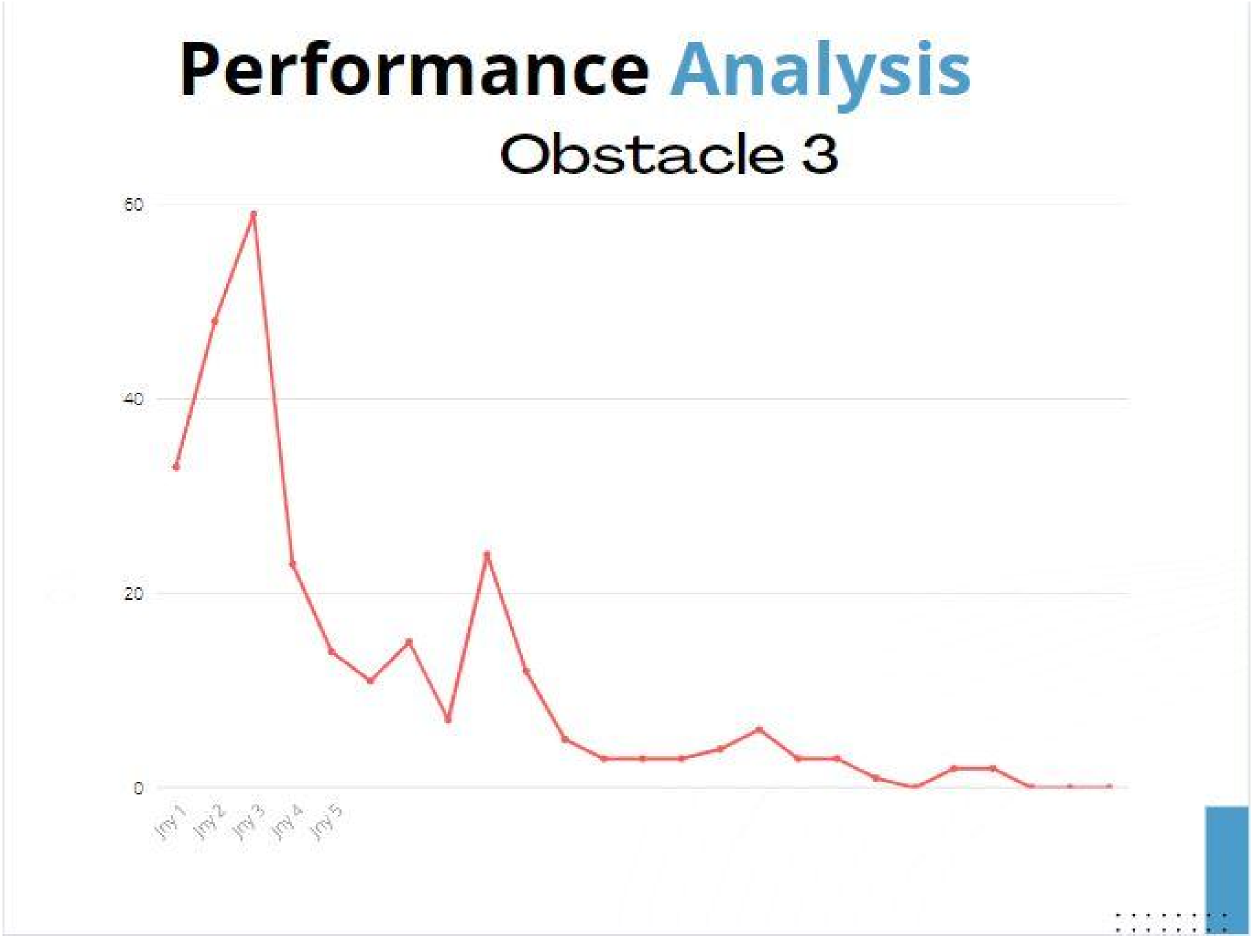

Figure 14 - Performance Analysis for Obstacle 3

Performance analysis is done for the environment in Obstacle-3(fig.9). First we can notice a lot of hits at the obstacle by the object . Once it learnt the environment , we can notice a drastic change in the no.of hits per journey.

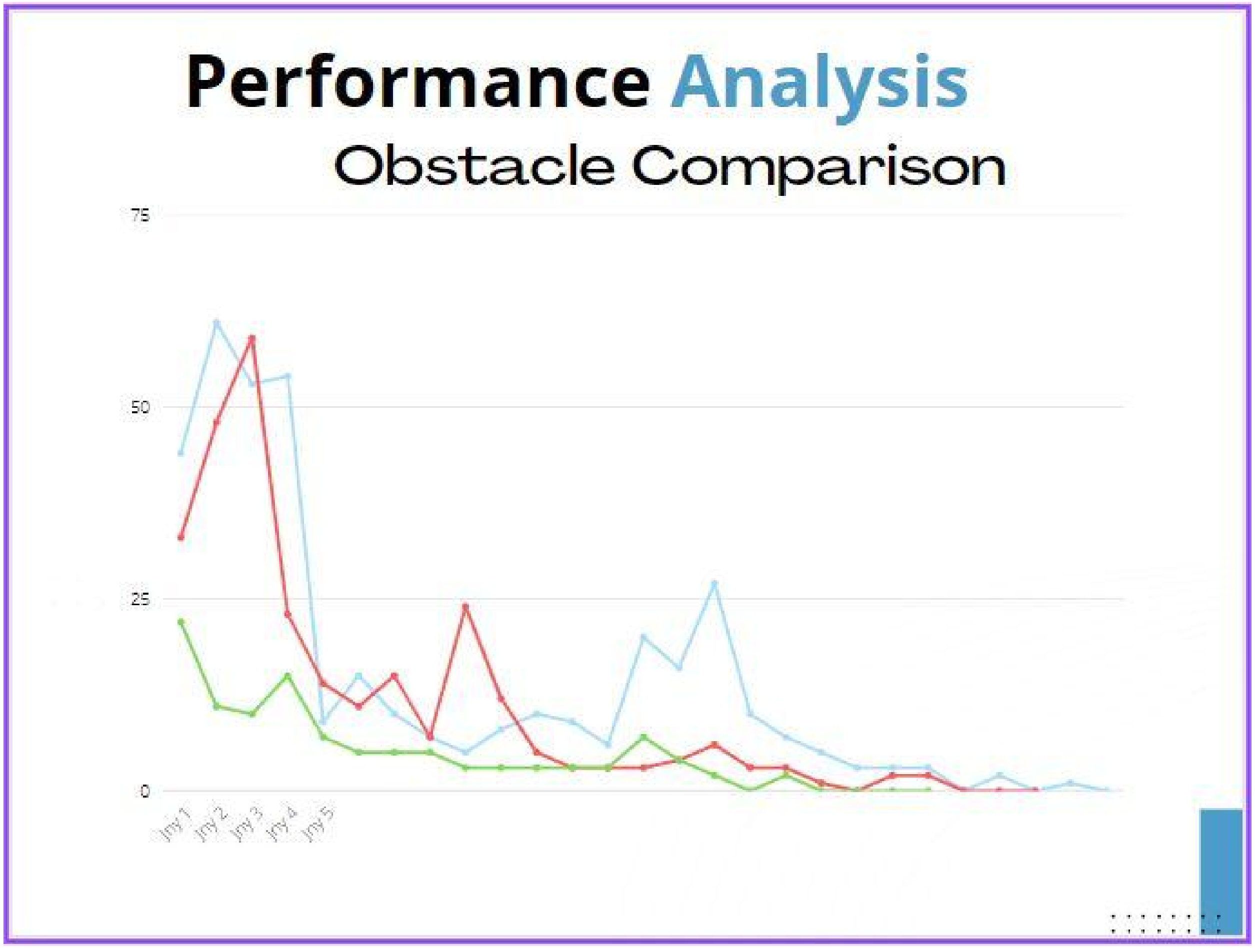

Figure 15 - Performance Analysis Comparison between all obstacle environments

Performance analysis comparison is done for all obstacle environments.This diagram depicts the complexity of each path. More the complexity ,more the number of hits at the obstacle at the beginning.

Conclusion :

Thus we have presented a deep Q-learning based model trained in a virtual environment that is able to make decisions for navigation in an adaptive way. As inputs it took the information from the three sensors and its current orientation. As output it decided the Q-values for each of the actions of going straight, turning left or turning right. As for the rewards, we punished it badly for hitting the sand, punished it slightly for going in the wrong direction and rewarded it slightly for going in the right direction.

Kivy was used to emulate the fire environment and PyTorch was used to communicate data and controls between the virtual environment and the deep learning model. The model was successfully able to navigate extreme fires based on its acquired knowledge and experience.

This work serves as the foundation on which to build a deep learning framework that is capable of identifying objects within the environment and incorporating those objects into its decision making process in order to successfully deliver safe, navigable routes to firefighters.

References :

[1] B. Jang, M. Kim, G. Harerimana and J. W. Kim, "Q-Learning Algorithms: A Comprehensive Classification and Applications," in *IEEE Access*, vol. 7, pp. 133653-133667, 2019, doi: 10.1109/ACCESS.2019.2941229.

[2] Jiang, L., Huang, H., & Ding, Z. (2019). Path planning for intelligent robots based on deep Q-learning with experience replay and heuristic knowledge. IEEE/CAA Journal of Automatica Sinica, 1–11. doi:10.1109/jas.2019.1911732

[3] L. Lv, S. Zhang, D. Ding and Y. Wang, "Path Planning via an Improved DQN-Based Learning Policy," in IEEE Access, vol. 7, pp. 67319-67330, 2019, doi: 10.1109/ACCESS.2019.2918703.

[4] S. Y. Luis, D. G. Reina and S. L. T. Marín, "A Multiagent Deep Reinforcement Learning Approach for Path Planning in Autonomous Surface Vehicles: The Ypacaraí Lake Patrolling Case," in *IEEE Access*, vol. 9, pp. 17084-17099, 2021, doi: 10.1109/ACCESS.2021.3053348.

[5] J. Liao, T. Liu, X. Tang, X. Mu, B. Huang and D. Cao, "Decision-Making Strategy on Highway for Autonomous Vehicles Using Deep Reinforcement Learning," in *IEEE Access*, vol. 8, pp. 177804-177814, 2020, doi: 10.1109/ACCESS.2020.3022755.

[6] S. Jiang, Z. Huang and Y. Ji, "Adaptive UAV-Assisted Geographic Routing With Q-Learning in VANET," in IEEE Communications Letters, vol. 25, no. 4, pp. 1358-1362, April 2021, doi: 10.1109/LCOMM.2020.3048250.

[7] Huang, R, Qin, C, Li, JL, Lan, X. Path planning of mobile robot in unknown dynamic continuous environment using reward-modified deep Q-network. Optim Control Appl Meth. , pp. 1– 18, 2021, https://doi.org/10.1002/oca.2781.

[8] Bae, Hyansu, Gidong Kim, Jonguk Kim, Dianwei Qian, and Sukgyu Lee. "Multi-Robot Path Planning Method Using Reinforcement Learning" Applied Sciences 9, pp no. 15: 3057 ,2019, https://doi.org/10.3390/app9153057.

[9] M. R. Lemos, A. V. R. de Souza, R. S. de Lira, C. A. O. de Freitas, V. J. da Silva and V. F. de Lucena, "Robot Training and Navigation through the Deep Q-Learning Algorithm," 2021 IEEE International Conference on Consumer

Electronics (ICCE), pp. 1-6, 2021, doi: 10.1109/ICCE50685.2021.9427675.

[10] Yan, C., Xiang, X. & Wang, C. Towards Real-Time Path Planning through Deep Reinforcement Learning for a UAV in Dynamic Environments. J Intell Robot Syst 98, pp. 297–309 , 2020, https://doi.org/10.1007/s10846-019-01073-3.

[11] W. Zaher, A. W. Youssef, L. A. Shihata, E. Azab and M. Mashaly, "Omnidirectional-Wheel Conveyor Path Planning and Sorting Using Reinforcement Learning Algorithms," in IEEE Access, vol. 10, pp. 27945-27959, 2022, doi: 10.1109/ACCESS.2022.3156924.

[12] Zhang, L., Liu, Z., Zhang, Y., and Ai, J. (2018). Intelligent path planning and following for uavs in forest surveillance and fire fighting missions. In 2018 IEEE CSAA Guidance, Navigation and Control Conference (CGNCC), pages 1–6. IEEE doi:https://doi.org/10.1007/s10586-021-03276-6.

[13] Ranaweera, D. M., Hemapala, K. U., Buddhika, A., and Jayasekara, P. (2018). A shortest path planning algorithm for pso base firefighting robots. In 2018 Fourth International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB), pages 1–5. IEEE. doi: https://doi.org/10.1109/AEEICB.2018.8480971

[14] Meyes, R., Tercan, H., Roggendorf, S., Thiele, T., B¨uscher, C., Obdenbusch, M., Brecher, C., Jeschke, S., and Meisen, T. (2017). Motion planning for industrial robots using reinforcement learning. Procedia CIRP, 63:107–112 doi: https://doi.org/10.1016/j.promfg.2020.01.023

[15] Jarvis, R. A. and Marzouqi, M. S. (2005). Robot path planning in high risk fire front environments. In TENCON 2005-2005 IEEE Region 10 Conference, pages 1–6. IEEE doi: https://doi.org/10.1109/LEOS.2005.1547854