Vatsa Nagaria
2019130041
Batch C
TE COMPS
Date - 11/10/21

Experiment-2

Aim: To build an intelligent agent that implements BFS and DFS.

PEAS:

Performance measure	Safety, Optimum speed, Comfortable journey
Environment	4-lane highway, surrounding vehicles
Actuators	Steering, Accelerator, Brakes
Sensors	Cameras

Code:

```
# environment
# the vehicle is on a multi-lane road and the position of the vehicle
# that is the lane in which the vehicle is will be the leftmost lane
# assuming it has just started for this environment the vehicles on
# the rear and sides are not considered and the vehicles in the front will be
considered.
# the agent will the root node and the vehicles just in front in each lane
# will be the next level which will be randomly generated and using bfs or dfs
# which gap if present between any vehicle in front should be taken is decided
# the level in front of the next level is not visible to the agent
# using bfs
# for the vehicles ahead we will find all the gaps and the nearest gap is
selected
# using dfs
# the first gap is visited


import turtle
import random
import time
```

```
windowGraphic = turtle.Screen()
windowGraphic.title("4- lane highway")
windowGraphic.setup(1000, 800)
windowGraphic.bgpic("road.png")
windowGraphic.listen()
windowGraphic.register_shape("car31.gif")
windowGraphic.register_shape("car32.gif")

myCar = turtle.Turtle()
myCar.hideturtle()
myCar.shape("car31.gif")
myCar.shapesize(1, 1)
myCar.up()
myCar.goto(-420, -250)
myCar.showturtle()


carsTop = []
carsMid = []
carsBot = []
x = -420
for i in range(4):
    car = turtle.Turtle()
    car.hideturtle()
    car.shape("car32.gif")
    car.shapesize(1, 1)
    car.up()
    car.goto(x, 250)
    carsTop.append(car)
    carm = car.clone()
    carm.goto(x, 0)
    carsMid.append(carm)
    carb = car.clone()
    carb.goto(x, -250)
    carsBot.append(carb)
    x = x + 280


def generateCarAhead():
    carArray = []
```

```
    for i in range(4):
        carArray.append(random.randint(0, 1))
    return carArray


myCarcords = 0
temp = 0
diff = 0
tempDiff = 0
tempNew = 0
newCarChords = 0
while(True):
    carPosition = generateCarAhead()
    if 0 in carPosition:
        m = 0
        for i in range(4):
            if(carPosition[i] == 0):
                temp = i
                if m == 0:
                    tempNew = temp
                    if newCarChords >= temp:
                        diff = newCarChords - temp
                    else:
                        diff = temp - newCarChords
                    m = 1
                if newCarChords >= temp:
                    tempDiff = newCarChords - temp
                else:
                    tempDiff = temp - newCarChords
                if tempDiff <= diff:
                    diff = tempDiff
                    tempNew = temp
        newCarChords = tempNew
        for i in range(4):
            if(carPosition[i] == 1):
                carsTop[i].showturtle()
        time.sleep(0.5)
        myCar.goto(-420+(newCarChords*280), 0)
        myCarcords = newCarChords
        for i in range(4):
```

```
            if(carPosition[i] == 1):
                carsTop[i].hideturtle()
                carsMid[i].showturtle()
        time.sleep(0.5)
        for i in range(4):
            if(carPosition[i] == 1):
                carsMid[i].hideturtle()
                carsBot[i].showturtle()
        time.sleep(0.5)
        for i in range(4):
            if(carPosition[i] == 1):
                carsBot[i].hideturtle()
        time.sleep(0.5)
        myCar.goto(-420+(myCarcords*280), -250)
    else:
        print("Waiting for lanes to get empty!!\n\n")
        for i in range(4):
            if(carPosition[i] == 1):
                carsTop[i].showturtle()
        time.sleep(1)
        for i in range(4):
            if(carPosition[i] == 1):
                carsTop[i].hideturtle()
        time.sleep(0.5)
```

Conclusion:

The environment for this experiment consists of a 4-lane highway and the surrounding vehicles, all the vehicles on either sides and rear are ignored, only the vehicles ahead of the agent are considered. The initial position of the vehicle is in the left lane assuming it has just started. The agent is considered as the root node and the empty lanes ahead as the child nodes. When all the lanes are full the agent does not change lanes and maintains the same speed moves ahead in the same lane. When there are empty lanes, using DFS the agent selects the first empty lane it finds and increases speed and changes lanes and continues in the same. Using BFS, the agent goes through all the empty lanes and selects the lane which has minimum cost i.e. which is the closest and would take minimum time to reach. The agent then increases speed and changes lane and continues in the same. This environment is restricted for only straight roads, uniform speed for other vehicles and 4 lanes in this case but can be done for any number of lanes also there is no case that covers accidents as the agent does not change lanes unless it finds an empty lane. The environment also assumes that the roads have a proper divider and the agent only detects vehicles and lanes on the side of the

divider where it is present. I implemented BFS algorithm for this agent. One challenge we faced in this experiment was how we could demonstrate our agent in such a way that would replicate its real life behaviour. We searched on the web and we came across Turtle which is a python library for creating pictures and shapes with the help of virtual canvas. We saw a lot of videos on how to use turtle and then we used turtle to demonstrate our agent in a much better way.