Nama Kelompok :

1. Javen Jonathan
2. Farhan Abie Ardandy
3. Muhammad Abubakar Rizvi
4. Rahmatul Fajri

Kelas : Khasanah Ilmi

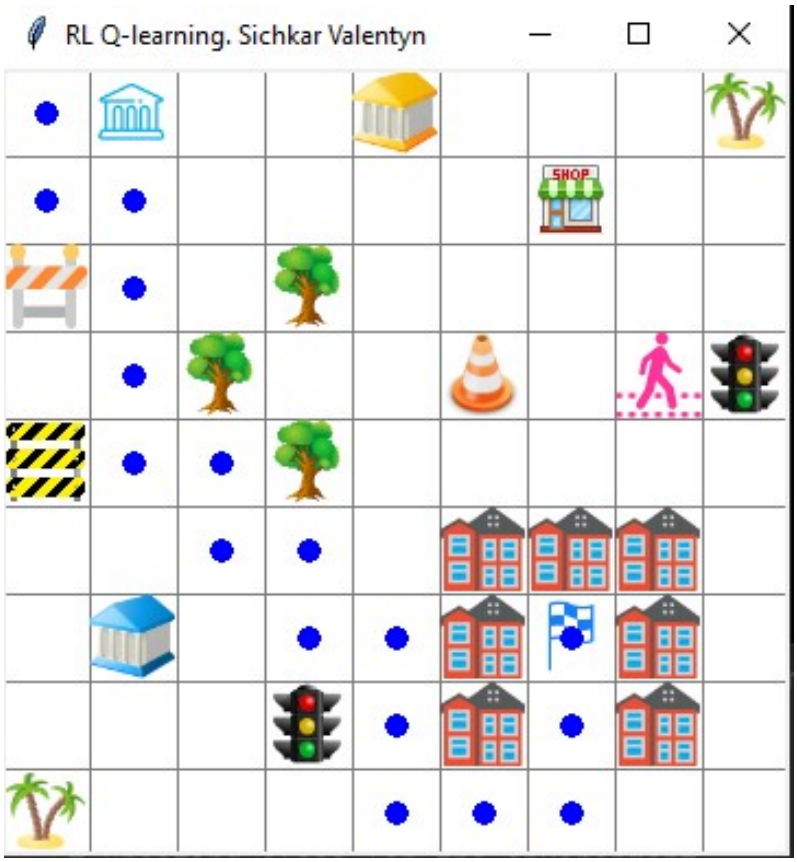
**TUGAS RL: Q-Learning Implementation**

- Terdapat 3 file dan 1 folder image, 3 file tersebut adalah agent_brain.py berisikan tentang algoritma QL, env.py berisikan tentang cara membuat environment yang mengambil input gambar dari folder image, dan run_agent.py yang berisikan untuk running agent.
- Untuk menjalankan program, tinggal jalankan run_agent.py saja.
- **Tugas:** merubah tata letak environment yang sudah ada, mulai dari posisi robot, posisi goal yang dituju, dan tata letak obstacle. Rubah di bagian env.py

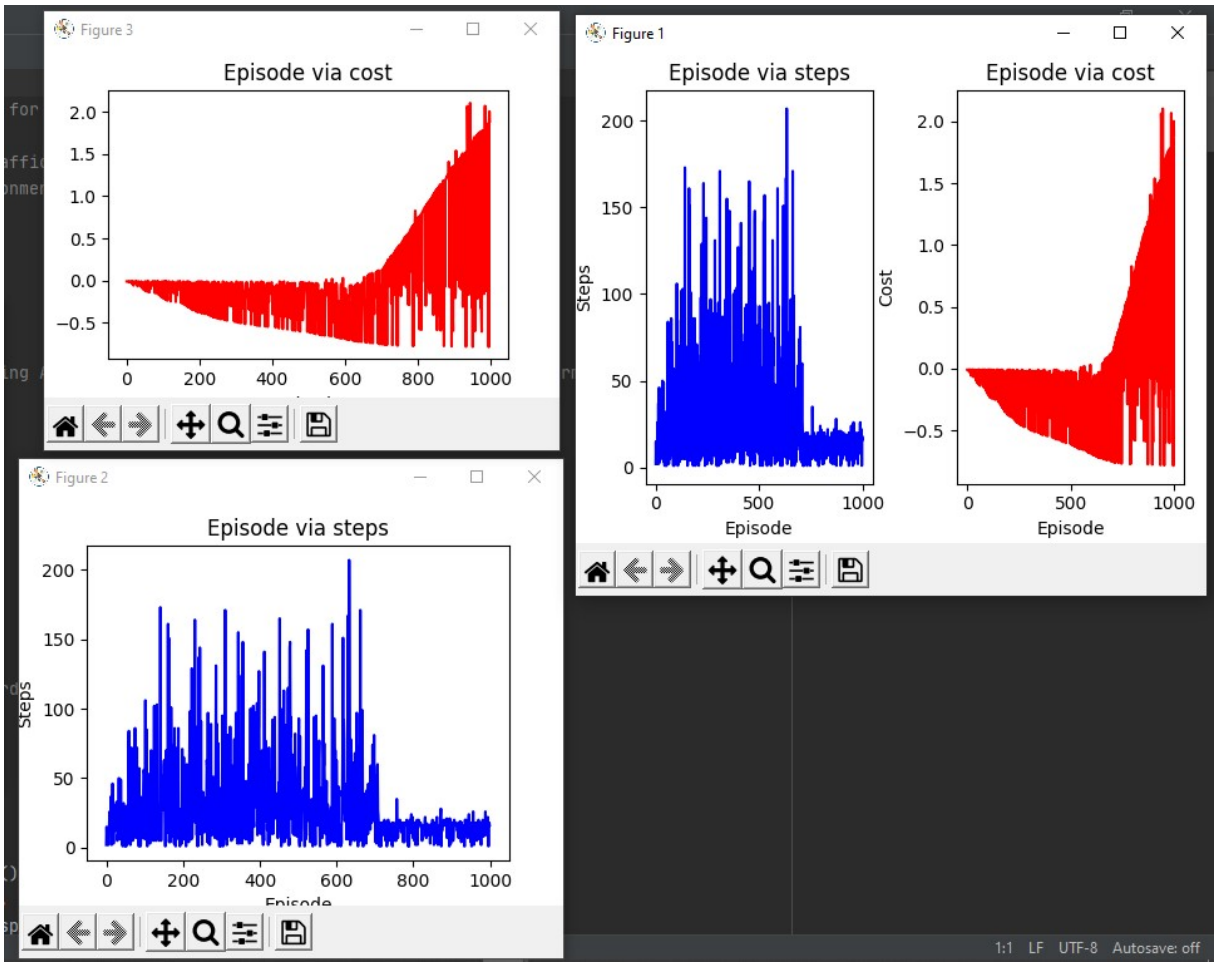
Code in Drive link:
https://ptorbitventurainodnesiamy.sharepoint.com/:f:/g/personal/ryan_orbitfutureacademy_sch_id/EvGMVJnTVY1FnBzGvWwY4SIBd30m1LhJ36Jk3Ychbl3Qow?e=nYEFav

---

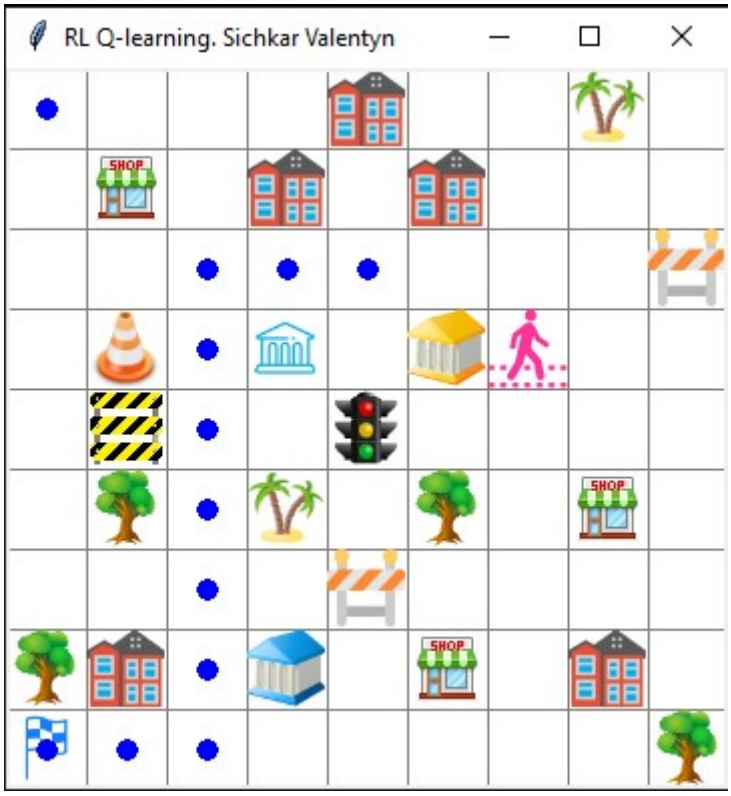
Tampilan awal run_agent.py (sebelum diubah):



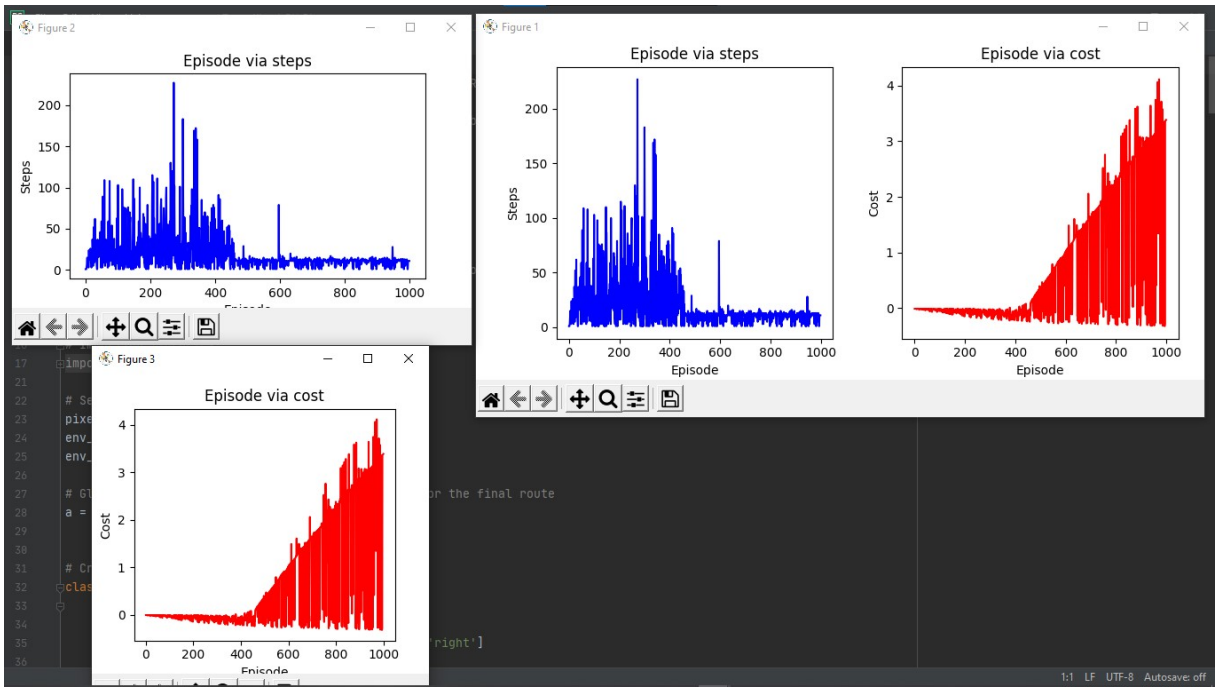
Grafik yang didapatkan:



Tampilan run_agent.py setelah diubah:




Grafik yang didapatkan:




## Perubahan:

- Penempatan final point diubah ke posisi (pixels * 0, pixels * 8)
- Penempatan agent diubah ke posisi (pixels * 4, pixels * 1)
- Penambahan dan perubahan tempat pada beberapa obstacle seperti pada gambar diatas.

Program:

```
# File: env.py
# Description: Building the environment-1 for the Mobile Robot to explore
# Agent - Mobile Robot
# Obstacles - 'road closed', 'trees', 'traffic lights', 'buildings'
# Environment: PyCharm and Anaconda environment
#
# MIT License
# Copyright (c) 2018 Valentyn N Sichkar
# github.com/sichkar-valentyn
#
# Reference to:
# Valentyn N Sichkar. Reinforcement Learning Algorithms for global path planning // GitHub platform. DOI: 10.5281/zenodo.1317899



# Importing libraries
import numpy as np  # To deal with data in form of matrices
import tkinter as tk  # To build GUI
import time  # Time is needed to slow down the agent and to see how he runs
from PIL import Image, ImageTk  # For adding images into the canvas widget

# Setting the sizes for the environment
pixels = 40   # pixels
env_height = 9  # grid height
env_width = 9  # grid width

# Global variable for dictionary with coordinates for the final route
a = {}



# Creating class for the environment
class Environment(tk.Tk, object):
    def __init__(self):
        super(Environment, self).__init__()
```

```
        self.action_space = ['up', 'down', 'left', 'right']
        self.n_actions = len(self.action_space)
        self.title('RL Q-learning. Sichkar Valentyn')
        self.geometry('{0}x{1}'.format(env_height * pixels, env_height * pixels))
        self.build_environment()

        # Dictionaries to draw the final route
        self.d = {}
        self.f = {}

        # Key for the dictionaries
        self.i = 0

        # Writing the final dictionary first time
        self.c = True

        # Showing the steps for longest found route
        self.longest = 0

        # Showing the steps for the shortest route
        self.shortest = 0

    # Function to build the environment
    def build_environment(self):
        self.canvas_widget = tk.Canvas(self,  bg='white',
                                       height=env_height * pixels,
                                       width=env_width * pixels)

        # Uploading an image for background
        # img_background = Image.open("images/bg.png")
        # self.background = ImageTk.PhotoImage(img_background)
        # # Creating background on the widget
        # self.bg = self.canvas_widget.create_image(0, 0, anchor='nw', image=self.background)

        # Creating grid lines
        for column in range(0, env_width * pixels, pixels):
            x0, y0, x1, y1 = column, 0, column, env_height * pixels
```

```
        self.canvas_widget.create_line(x0, y0, x1, y1, fill='grey')
    for row in range(0, env_height * pixels, pixels):
        x0, y0, x1, y1 = 0, row, env_height * pixels, row
        self.canvas_widget.create_line(x0, y0, x1, y1, fill='grey')

    # Creating objects of  Obstacles
    # Obstacle type 1 - road closed1
    img_obstacle1 = Image.open("images/road_closed1.png")
    self.obstacle1_object = ImageTk.PhotoImage(img_obstacle1)
    # Obstacle type 2 - tree1
    img_obstacle2 = Image.open("images/tree1.png")
    self.obstacle2_object = ImageTk.PhotoImage(img_obstacle2)
    # Obstacle type 3 - tree2
    img_obstacle3 = Image.open("images/tree2.png")
    self.obstacle3_object = ImageTk.PhotoImage(img_obstacle3)
    # Obstacle type 4 - building1
    img_obstacle4 = Image.open("images/building1.png")
    self.obstacle4_object = ImageTk.PhotoImage(img_obstacle4)
    # Obstacle type 5 - building2
    img_obstacle5 = Image.open("images/building2.png")
    self.obstacle5_object = ImageTk.PhotoImage(img_obstacle5)
    # Obstacle type 6 - road closed2
    img_obstacle6 = Image.open("images/road_closed2.png")
    self.obstacle6_object = ImageTk.PhotoImage(img_obstacle6)
    # Obstacle type 7 - road closed3
    img_obstacle7 = Image.open("images/road_closed3.png")
    self.obstacle7_object = ImageTk.PhotoImage(img_obstacle7)
    # Obstacle type 8 - traffic lights
    img_obstacle8 = Image.open("images/traffic_lights.png")
    self.obstacle8_object = ImageTk.PhotoImage(img_obstacle8)
    # Obstacle type 9 - pedestrian
    img_obstacle9 = Image.open("images/pedestrian.png")
    self.obstacle9_object = ImageTk.PhotoImage(img_obstacle9)
    # Obstacle type 10 - shop
    img_obstacle10 = Image.open("images/shop.png")
    self.obstacle10_object = ImageTk.PhotoImage(img_obstacle10)
    # Obstacle type 11 - bank1
```

```
        img_obstacle11 = Image.open("images/bank1.png")
        self.obstacle11_object = ImageTk.PhotoImage(img_obstacle11)
        # Obstacle type 12 - bank2
        img_obstacle12 = Image.open("images/bank2.png")
        self.obstacle12_object = ImageTk.PhotoImage(img_obstacle12)

        # Creating obstacles themselves
        # Obstacles from 1 to 22
        self.obstacle1 = self.canvas_widget.create_image(pixels * 0, pixels * 7, anchor='nw',
image=self.obstacle2_object)
        # Obstacle 2
        self.obstacle2 = self.canvas_widget.create_image(pixels * 8, pixels * 2, anchor='nw',
image=self.obstacle6_object)
        # Obstacle 3
        self.obstacle3 = self.canvas_widget.create_image(pixels * 3, pixels * 3, anchor='nw',
image=self.obstacle5_object)
        # Obstacle 4
        self.obstacle4 = self.canvas_widget.create_image(pixels * 4, pixels * 4, anchor='nw',
image=self.obstacle8_object)
        # Obstacle 5
        self.obstacle5 = self.canvas_widget.create_image(pixels * 5, pixels * 5, anchor='nw',
image=self.obstacle2_object)
        # Obstacle 6
        self.obstacle6 = self.canvas_widget.create_image(pixels * 1, pixels * 3, anchor='nw',
image=self.obstacle7_object)
        # Obstacle 7
        self.obstacle7 = self.canvas_widget.create_image(pixels * 6, pixels * 3, anchor='nw',
image=self.obstacle9_object)
        # Obstacle 8
        self.obstacle8 = self.canvas_widget.create_image(pixels * 7, pixels * 5, anchor='nw',
image=self.obstacle10_object)
        # Obstacle 9
        self.obstacle9 = self.canvas_widget.create_image(pixels * 1, pixels * 7, anchor='nw',
image=self.obstacle4_object)
        # Obstacle 10
        self.obstacle10 = self.canvas_widget.create_image(pixels * 8, pixels * 8, anchor='nw',
image=self.obstacle2_object)
```

```
        # Obstacle 11
        self.obstacle11 = self.canvas_widget.create_image(pixels * 4, pixels * 0, anchor='nw',
image=self.obstacle4_object)
        # Obstacle 12
        self.obstacle12 = self.canvas_widget.create_image(pixels * 4, pixels * 6, anchor='nw',
image=self.obstacle6_object)
        # Obstacle 13
        self.obstacle13 = self.canvas_widget.create_image(pixels *3, pixels * 5, anchor='nw',
image=self.obstacle3_object)
        # Obstacle 14
        self.obstacle14 = self.canvas_widget.create_image(pixels * 5, pixels * 7, anchor='nw',
image=self.obstacle10_object)
        # Obstacle 15
        self.obstacle15 = self.canvas_widget.create_image(pixels * 1, pixels * 4, anchor='nw',
image=self.obstacle1_object)
        # Obstacle 16
        self.obstacle16 = self.canvas_widget.create_image(pixels * 7, 0, anchor='nw',
image=self.obstacle3_object)
        # Obstacle 17
        self.obstacle17 = self.canvas_widget.create_image(pixels * 5, pixels * 1, anchor='nw',
image=self.obstacle4_object)
        # Obstacle 18
        self.obstacle18 = self.canvas_widget.create_image(pixels * 3, pixels * 7, anchor='nw',
image=self.obstacle11_object)
        # Obstacle 19
        self.obstacle19 = self.canvas_widget.create_image(pixels * 5, pixels * 3, anchor='nw',
image=self.obstacle12_object)
        # Obstacle 20
        self.obstacle20 = self.canvas_widget.create_image(pixels * 7, pixels * 7, anchor='nw',
image=self.obstacle4_object)
        # Obstacle 21
        self.obstacle21 = self.canvas_widget.create_image(pixels * 3, pixels * 1, anchor='nw',
image=self.obstacle4_object)
        # Obstacle 22
        self.obstacle22 = self.canvas_widget.create_image(pixels * 1, pixels * 5, anchor='nw',
image=self.obstacle2_object)
        # Obstacle 23
```

```
        self.obstacle23 = self.canvas_widget.create_image(pixels * 1, pixels * 1, anchor='nw',
image=self.obstacle10_object)

        # Final Point
        img_flag = Image.open("images/flag.png")
        self.flag_object = ImageTk.PhotoImage(img_flag)
        self.flag = self.canvas_widget.create_image(pixels * 0, pixels * 8, anchor='nw',
image=self.flag_object)

        # Uploading the image of Mobile Robot
        img_robot = Image.open("images/agent1.png")
        self.robot = ImageTk.PhotoImage(img_robot)

        # Creating an agent with photo of Mobile Robot
        self.agent = self.canvas_widget.create_image(pixels *4, pixels * 1, anchor='nw', image=self.robot)

        # Packing everything
        self.canvas_widget.pack()

    # Function to reset the environment and start new Episode
    def reset(self):
        self.update()
        # time.sleep(1)

        # Updating agentw
        self.canvas_widget.delete(self.agent)
        self.agent = self.canvas_widget.create_image(pixels* 4, pixels *1, anchor='nw', image=self.robot)

        # # Clearing the dictionary and the i
        self.d = {}
        self.i = 0

        # Return observation
        return self.canvas_widget.coords(self.agent)

    # Function to get the next observation and reward by doing next step
    def step(self, action):
```

```
    # Current state of the agent
    state = self.canvas_widget.coords(self.agent)
    base_action = np.array([0,0])

    # Updating next state according to the action
    # Action 'up'
    if action == 0:
        if state[1] >= pixels:
            base_action[1] -= pixels
    # Action 'down'
    elif action == 1:
        if state[1] < (env_height - 1) * pixels:
            base_action[1] += pixels
    # Action right
    elif action == 2:
        if state[0] < (env_width - 1) * pixels:
            base_action[0] += pixels
    # Action left
    elif action == 3:
        if state[0] >= pixels:
            base_action[0] -= pixels

    # Moving the agent according to the action
    self.canvas_widget.move(self.agent, base_action[0], base_action[1])

    # Writing in the dictionary coordinates of found route
    self.d[self.i] = self.canvas_widget.coords(self.agent)

    # Updating next state
    next_state = self.d[self.i]

    # Updating key for the dictionary
    self.i += 1

    # Calculating the reward for the agent
    if next_state == self.canvas_widget.coords(self.flag):
        reward = 1
```

```
            done = True
            next_state = 'goal'

            # Filling the dictionary first time
            if self.c == True:
                for j in range(len(self.d)):
                    self.f[j] = self.d[j]
                self.c = False
                self.longest = len(self.d)
                self.shortest = len(self.d)

            # Checking if the currently found route is shorter
            if len(self.d) < len(self.f):
                # Saving the number of steps for the shortest route
                self.shortest = len(self.d)
                # Clearing the dictionary for the final route
                self.f = {}
                # Reassigning the dictionary
                for j in range(len(self.d)):
                    self.f[j] = self.d[j]

            # Saving the number of steps for the longest route
            if len(self.d) > self.longest:
                self.longest = len(self.d)

        elif next_state in [self.canvas_widget.coords(self.obstacle1),
                            self.canvas_widget.coords(self.obstacle2),
                            self.canvas_widget.coords(self.obstacle3),
                            self.canvas_widget.coords(self.obstacle4),
                            self.canvas_widget.coords(self.obstacle5),
                            self.canvas_widget.coords(self.obstacle6),
                            self.canvas_widget.coords(self.obstacle7),
                            self.canvas_widget.coords(self.obstacle8),
                            self.canvas_widget.coords(self.obstacle9),
                            self.canvas_widget.coords(self.obstacle10),
                            self.canvas_widget.coords(self.obstacle11),
                            self.canvas_widget.coords(self.obstacle12),
```

```
                    self.canvas_widget.coords(self.obstacle13),
                    self.canvas_widget.coords(self.obstacle14),
                    self.canvas_widget.coords(self.obstacle15),
                    self.canvas_widget.coords(self.obstacle16),
                    self.canvas_widget.coords(self.obstacle17),
                    self.canvas_widget.coords(self.obstacle18),
                    self.canvas_widget.coords(self.obstacle19),
                    self.canvas_widget.coords(self.obstacle20),
                    self.canvas_widget.coords(self.obstacle21),
                    self.canvas_widget.coords(self.obstacle22),
                    self.canvas_widget.coords(self.obstacle23)]:
            reward = -1
            done = True
            next_state = 'obstacle'

            # Clearing the dictionary and the i
            self.d = {}
            self.i = 0

        else:
            reward = 0
            done = False

        return next_state, reward, done

    # Function to refresh the environment
    def render(self):
        #time.sleep(0.03)
        self.update()

    # Function to show the found route
    def final(self):
        # Deleting the agent at the end
        self.canvas_widget.delete(self.agent)

        # Showing the number of steps
        print('The shortest route:', self.shortest)
```

```
        print('The longest route:', self.longest)

        # Creating initial point
        origin = np.array([20, 20])
        self.initial_point = self.canvas_widget.create_oval(
            origin[0] - 5, origin[1] - 5,
            origin[0] + 5, origin[1] + 5,
            fill='blue', outline='blue')

        # Filling the route
        for j in range(len(self.f)):
            # Showing the coordinates of the final route
            print(self.f[j])
            self.track = self.canvas_widget.create_oval(
                self.f[j][0] + origin[0] - 5, self.f[j][1] + origin[0] - 5,
                self.f[j][0] + origin[0] + 5, self.f[j][1] + origin[0] + 5,
                fill='blue', outline='blue')
            # Writing the final route in the global variable a
            a[j] = self.f[j]



# Returning the final dictionary with route coordinates
# Then it will be used in agent_brain.py
def final_states():
    return a



# This we need to debug the environment
# If we want to run and see the environment without running full algorithm
if __name__ == '__main__':
    env = Environment()
    env.mainloop()
```