KULLIYYAH OF INFORMATION AND COMMUNICATION TECHNOLOGY

CSCI 2304 – INTELLIGENT SYSTEMS

SEMESTER I 2021/2022

Game Title: TurtleWay! (Maze Solver)

Source code link: [https://github.com/kinah00/Assignment-IS.git]

LECTURER: ASSOC. PROF. TS. DR. AMELIA RITAHANI BINTI ISMAIL

GROUP MEMBERS:

NAME	MATRIC NUMBER
SAKINAH BINTI SHAMSUDDIN	1911912
NURUL HANIS BINTI MOHD DHUZUKI	1913364

Section: 02

Submission Date:

11th November 2021

CONTENTS

1. OVERVIEW

1.1. Background

Plastic straw and climate changes. These words are no longer unusual to us. Climate changes is exposing sea turtles to threats and putting food production at risk. In the warming ocean, a strong female turtle named Timah, must find its way to find some food to survive.

One day, Timah felt starving. She knew that she had to consume some energy because plastic straws would not do anything to her. Out of the sudden, she got stuck in between the coral reefs. Through the bleaching coral reefs, she tried to free herself. From the eagle eye view, the barriers look was just like a maze!

Starvation did not stop her persistency. In the presence of Graph Theory and Search Algorithm, this smart turtle discovered an algorithm called Breadth-first Search to solve her maze problem. What an *ocean-genius*! She attempted to draft some nodes and edges on the sand, hoping for a precious and scrumptious plate of seaweed, she decided to find the shortest path to eat. The starting node was labelled red while the end node is green.

In the end, she found her way. Now, every ocean creature has known her from her intelligence. Her fame led to human acknowledgement. She had an interview for her journey and two women, the programmers believed that her story could be an interesting game to be coded in Python. Therefore, here comes this maze game, *TurtleWay!*

1.2. Game Flow

The player will play as Timah the Turtle. The flow of the game as shown below.

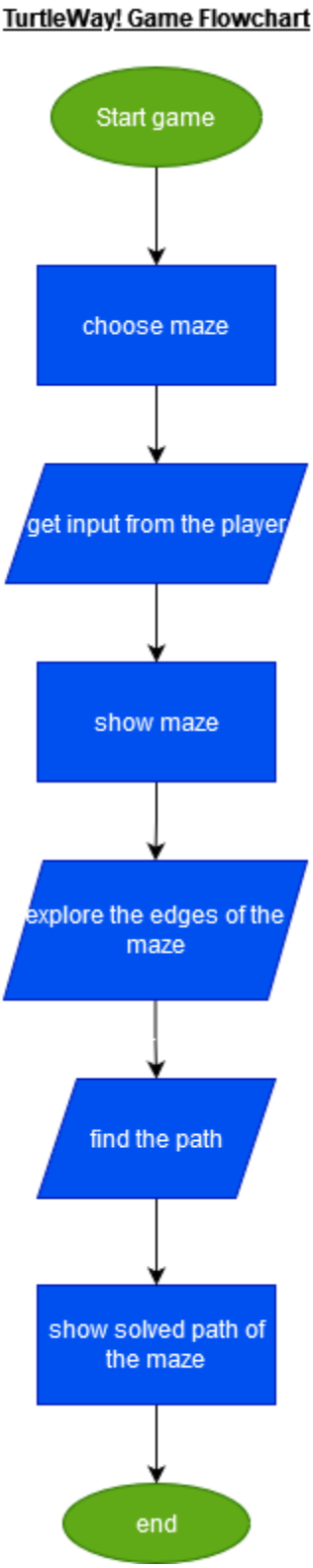

Figure 1: Game Flowchart.

2. GRAPH THEORY AND SEARCH ALGORTIHM APPROACHES

2.1. Implementation

Graph Theory and Breadth-first Search are chosen to be implemented into this game. A traversing algorithm like Breadth-first Search selected the starting node to start, to the next neighbours by traversing the layer by layer, then the defined target node. This algorithm is easy to see in queue to determine the visited or unvisited nodes to find the shortest path.

In solving the maze, each room will be represented into vertices while the doors are the edges. The starting and the end points were already stated in this maze game to make it easier. The movement follows to another, the vertices can navigate to its neighbour, the edges of the maze were explored or visited. Consequently, the path created when the edges are connecting the vertices. In this case, the newly discovered room is going to be in the queue. Every room that was visited and connected by the doors are stored in a queue, if the queue is empty then it is visited.

2.2. Code Structures - Python

The structures mentioned are based on the source code given.

i. Import

```
import turtle as trt                    #import turtle library
import random                           #to get random maze
import time                             #use time clock
import sys                              #import sys module
from collections import deque           #from collections module import deque(double ended queue)
import numpy as np                      #import numpy
```

ii. Lists

```
walls = []
path = []
visited = set()
frontier = deque()
solution = {}                                       # dictionary to store solution path
```

iii. Classes

```
maze = Maze()
red = StartRed()
purple = Check()
blue = PathWay()
yellow = RightPath()
```

Classes are used to define a particular type of object. A set of data of maze, starting point, visited grid, paths are as follows:

```
class Maze(trt.Turtle):
    def __init__(self):                 #create function with __init__ method that known as constructor
        trt.Turtle.__init__(self)       #use self to represents the instance of the class
        self.shape("square")            #change the turtle shape to square
        self.color("black")             #set the colour of the maze to black
        self.penup()                    #to make sure the turtle move without leaving trace
        self.speed(30)                  #set the speed of the turtle
```

```
#class for the green turtle
class PathWay(trt.Turtle):
    def __init__(self):                 #create function with __init__ method that known as constructor
        trt.Turtle.__init__(self)       #use self to represents the instance of the class
        self.shape("square")            #change the turtle shape to square
        self.color("blue")              #set the colour of the shape to blue
        self.penup()                    #to make sure the turtle move without leaving trace
        self.speed(30)                  #set the speed of the turtle
```

```
#class for the check path
class Check(trt.Turtle):
    def __init__(self):
        trt.Turtle.__init__(self)       #use self to represents the instance of the class
        self.shape("square")            #change the turtle shape to square
        self.color("purple")            #set the colour of the shape to purple
        self.penup()                    #to make sure the turtle move without leaving trace
        self.speed(30)                  #set the speed of the turtle
```

```
#class for the start point
class StartRed(trt.Turtle):
    def __init__(self):
        trt.Turtle.__init__(self)       #use self to represents the instance of the class
        self.shape("circle")            #change the turtle shape to circle
        self.color("red")               #set the colour of the shape to red
        self.penup()                    #to make sure the turtle move without leaving trace
        self.speed(10)                  #set the speed of the turtle
```

```
#class for the right path
class RightPath(trt.Turtle):
    def __init__(self):
        trt.Turtle.__init__(self)       #use self to represents the instance of the class
        self.shape("turtle")            #change the turtle shape to turtle shape
        self.color("yellow")            #set the colour of the turtle to yellow
        self.penup()                    #to make sure the turtle move without leaving trace
        self.speed(10)                  #set the speed of the turtle
```

iv. Arrays

Array shown below is used to store different types of maze for in player's option,

designMaze = np.array([[….], […], […]])

v. Functions

A function only runs when it is called.

```
def __init__(self):                 #create function with __init__ method that known as constructor
    trt.Turtle.__init__(self)       #use self to represents the instance of the class
    self.shape("square")            #change the turtle shape to square
    self.color("black")             #set the colour of the maze to black
    self.penup()                    #to make sure the turtle move without leaving trace
    self.speed(30)                  #set the speed of the turtle
```

```
def __init__(self):                 #create function with __init__ method that known as constructor
    trt.Turtle.__init__(self)       #use self to represents the instance of the class
    self.shape("square")            #change the turtle shape to square
    self.color("blue")              #set the colour of the shape to blue
    self.penup()                    #to make sure the turtle move without leaving trace
    self.speed(30)                  #set the speed of the turtle
```

```
def __init__(self):
    trt.Turtle.__init__(self)       #use self to represents the instance of the class
    self.shape("square")            #change the turtle shape to square
    self.color("purple")            #set the colour of the shape to purple
    self.penup()                    #to make sure the turtle move without leaving trace
    self.speed(30)                  #set the speed of the turtle
```

```
def __init__(self):
    trt.Turtle.__init__(self)       #use self to represents the instance of the class
    self.shape("circle")            #change the turtle shape to circle
    self.color("red")               #set the colour of the shape to red
    self.penup()                    #to make sure the turtle move without leaving trace
    self.speed(10)                  #set the speed of the turtle
```

```
def __init__(self):
    trt.Turtle.__init__(self)       #use self to represents the instance of the class
    self.shape("turtle")            #change the turtle shape to turtle shape
    self.color("yellow")            #set the colour of the turtle to yellow
    self.penup()                    #to make sure the turtle move without leaving trace
    self.speed(10)                  #set the speed of the turtle
```

```
def setup_maze(grid):                                   # define a function called setup_maze
    global start_x, start_y, end_x, end_y               # set up global variables for start and end locations
    for y in range(len(grid)):                          # read in the grid line by line
        for x in range(len(grid[y])):                   # read each cell in the line
            character = grid[y][x]                      # assign the varaible "character" the the x and y location of the grid
            screen_x = -588 + (x * 24)                  # move to the x location on the screen staring at -588
            screen_y = 288 - (y * 24)                   # move to the y location of the screen starting at 288

            if character == "+":
                maze.goto(screen_x, screen_y)           # move pen to the x and y location
                maze.stamp()                            # stamp a copy of the turtle on the screen
                walls.append((screen_x, screen_y))      # add coordinate to walls list

            if character == " " or character == "e":
                path.append((screen_x, screen_y))       # add " " and e to path list

            if character == "e":
                blue.color("green")
                blue.goto(screen_x, screen_y)
                end_x, end_y = screen_x,screen_y        # assign end locations variables to end_x and end_y
                blue.stamp()
                blue.color("blue")

            if character == "s":
                start_x, start_y = screen_x, screen_y   # assign start locations variables to start_x and start_y
                red.goto(screen_x, screen_y)
```

```
def search(x,y):
    frontier.append((x, y))
    solution[x,y] = x,y

    while len(frontier) > 0:                            # exit while loop when frontier queue equals zero
        time.sleep(0)
        x, y = frontier.popleft()                       # pop next entry in the frontier queue an assign to x and y location

        if(x - 24, y) in path and (x - 24, y) not in visited:  # check the cell on the left
            cell = (x - 24, y)
            solution[cell] = x, y                       # backtracking routine [cell] is the previous cell. x, y is the current cell
            purple.goto(cell)                           # identify frontier cells
            purple.stamp()
            frontier.append(cell)                       # add cell to frontier list
            visited.add((x-24, y))                      # add cell to visited list

        if (x, y - 24) in path and (x, y - 24) not in visited:  # check the cell down
            cell = (x, y - 24)
            solution[cell] = x, y
            purple.goto(cell)
            purple.stamp()
            frontier.append(cell)
            visited.add((x, y - 24))
            print(solution)

        if(x + 24, y) in path and (x + 24, y) not in visited:   # check the cell on the  right
            cell = (x + 24, y)
            solution[cell] = x, y
            purple.goto(cell)
            purple.stamp()
            frontier.append(cell)
            visited.add((x +24, y))

        if(x, y + 24) in path and (x, y + 24) not in visited:  # check the cell up
            cell = (x, y + 24)
            solution[cell] = x, y
            purple.goto(cell)
            purple.stamp()
```

```
def backRoute(x, y):
    yellow.goto(x, y)
    yellow.stamp()
    while (x, y) != (start_x, start_y):                # stop loop when current cells == start cell
        yellow.goto(solution[x, y])                    # move the yellow sprite to the key value of solution ()
        yellow.stamp()
        x, y = solution[x, y]                          # "key value" now becomes the new key
```

```
# main program starts here #
gridchoice = UserInput()
grid = designMaze[gridchoice]
setup_maze(grid)
search(start_x,start_y)
backRoute(end_x, end_y)
gameScreen.exitonclick()
```

Parameter can be passed into the function and a function can return data as result.

vi. Loops

- while loop

```
while len(frontier) > 0:                           # exit while loop when frontier queue equals zero
    time.sleep(0)
    x, y = frontier.popleft()                      # pop next entry in the frontier queue an assign to x and y location
```

- for loop

```
def setup_maze(grid):                              # define a function called setup_maze
    global start_x, start_y, end_x, end_y          # set up global variables for start and end locations
    for y in range(len(grid)):                     # read in the grid line by line
        for x in range(len(grid[y])):              # read each cell in the line
            character = grid[y][x]                 # assign the varaible "character" the the x and y location od the grid
            screen_x = -588 + (x * 24)             # move to the x location on the screen staring at -588
            screen_y = 288 - (y * 24)              # move to the y location of the screen starting at 288

            if character == "+":
                maze.goto(screen_x, screen_y)      # move pen to the x and y location and
                maze.stamp()                       # stamp a copy of the turtle on the screen
                walls.append((screen_x, screen_y)) # add coordinate to walls list

            if character == " " or character == "e":
```

- Conditions and If statement

```
if(x - 24, y) in path and (x - 24, y) not in visited:  # check the cell on the left
    cell = (x - 24, y)
    solution[cell] = x, y                    # backtracking routine [cell] is the previous cell. x, y is the current cell
    purple.goto(cell)                        # identify frontier cells
    purple.stamp()
    frontier.append(cell)                    # add cell to frontier list
    visited.add((x-24, y))                   # add cell to visited list

if (x, y - 24) in path and (x, y - 24) not in visited:  # check the cell down
    cell = (x, y - 24)
    solution[cell] = x, y
    purple.goto(cell)
    purple.stamp()
    frontier.append(cell)
    visited.add((x, y - 24))
    print(solution)

if(x + 24, y) in path and (x + 24, y) not in visited:   # check the cell on the  right
    cell = (x + 24, y)
    solution[cell] = x, y
    purple.goto(cell)
    purple.stamp()
    frontier.append(cell)
    visited.add((x +24, y))

if(x, y + 24) in path and (x, y + 24) not in visited:  # check the cell up
    cell = (x, y + 24)
    solution[cell] = x, y
    purple.goto(cell)
    purple.stamp()
    frontier.append(cell)
    visited.add((x, y + 24))
blue.goto(x,y)
blue.stamp()
```

3. REFERENCES

1. M. (2020, August 18). *Understanding self and __init__ method in python Class.* MicroPyramid. https://micropyramid.com/blog/understand-self-and-__init__-method-in-python-class/
2. Real Python. (2021, June 26). *The Beginner's Guide to Python Turtle*. https://realpython.com/beginners-guide-python-turtle/
3. *Python Dictionaries*. W3schools. https://www.w3schools.com/python/python_dictionaries.asp
4. Soularidis, A. (2021, August 17). *Solve Maze using Breadth-First Search (BFS) algorithm in Python*. Medium. https://medium.com/nerd-for-tech/solve-maze-using-breadth-first-search-bfs-algorithm-in-python-7931acbe8a93