NAME – BHARGAVA NUTHAKKI

CS-470 ARTIFICIAL INTELLIGENCE

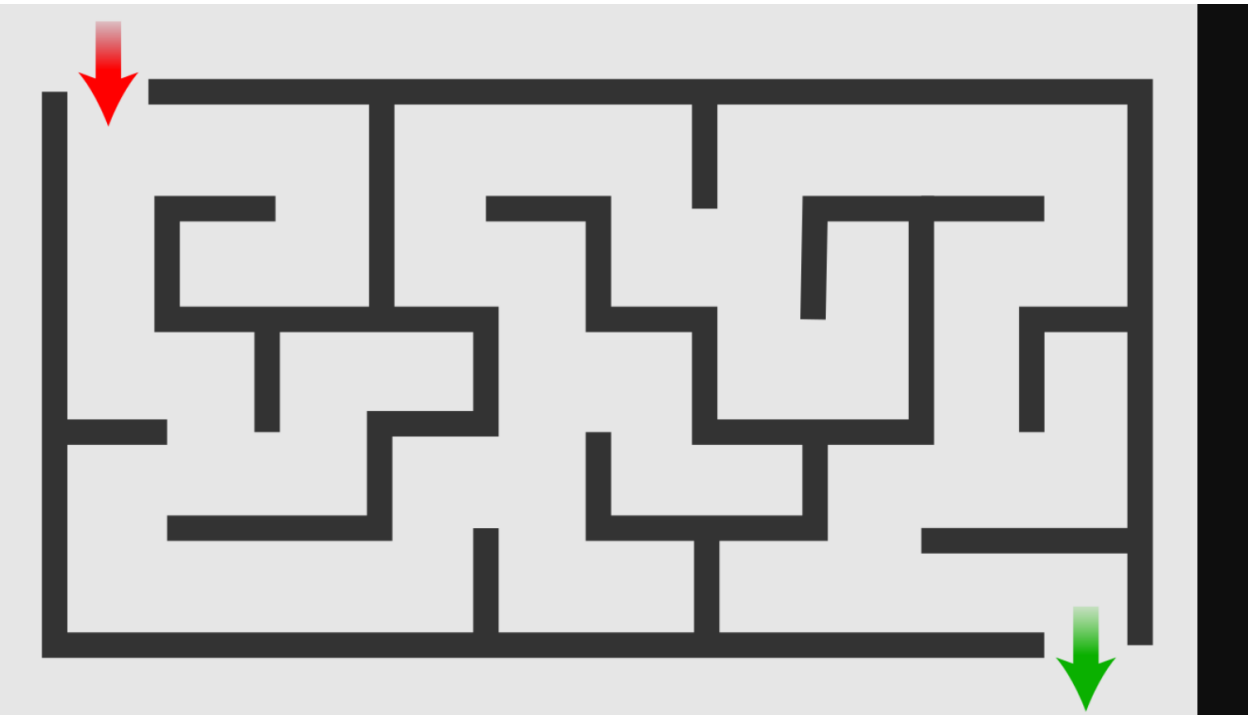
FINAL PROJECT

PROJECT NAME – MAZE RUNNER

DATE OF SUBMISSION – 12/08/2021

- **CODE AND OUTPUT OF SIMPLE MAZE FILE.**

INPUT-



CODE-

```
import turtle                    # import turtle library
import time
import sys
from collections import deque

wn = turtle.Screen()               # define the turtle screen
wn.bgcolor("white")                # set the background colour
wn.title("Breadth First Search Maze Solving Program")
wn.setup(width=1.0,height=1.0)                # setup the dimensions of the
working window


# this is the class for the Maze
class Maze(turtle.Turtle):                # define a Maze class
    def __init__(self):
        turtle.Turtle.__init__(self)
        self.shape("square")             # the turtle shape
        self.color("black")
        self.shapesize(0.5)              # colour of the turtle
        self.penup()                     # lift up the pen so it do not leave
a trail
        self.speed(0)

# this is the class for the finish line - green square in the maze
class Cyan(turtle.Turtle):
    def __init__(self):
```

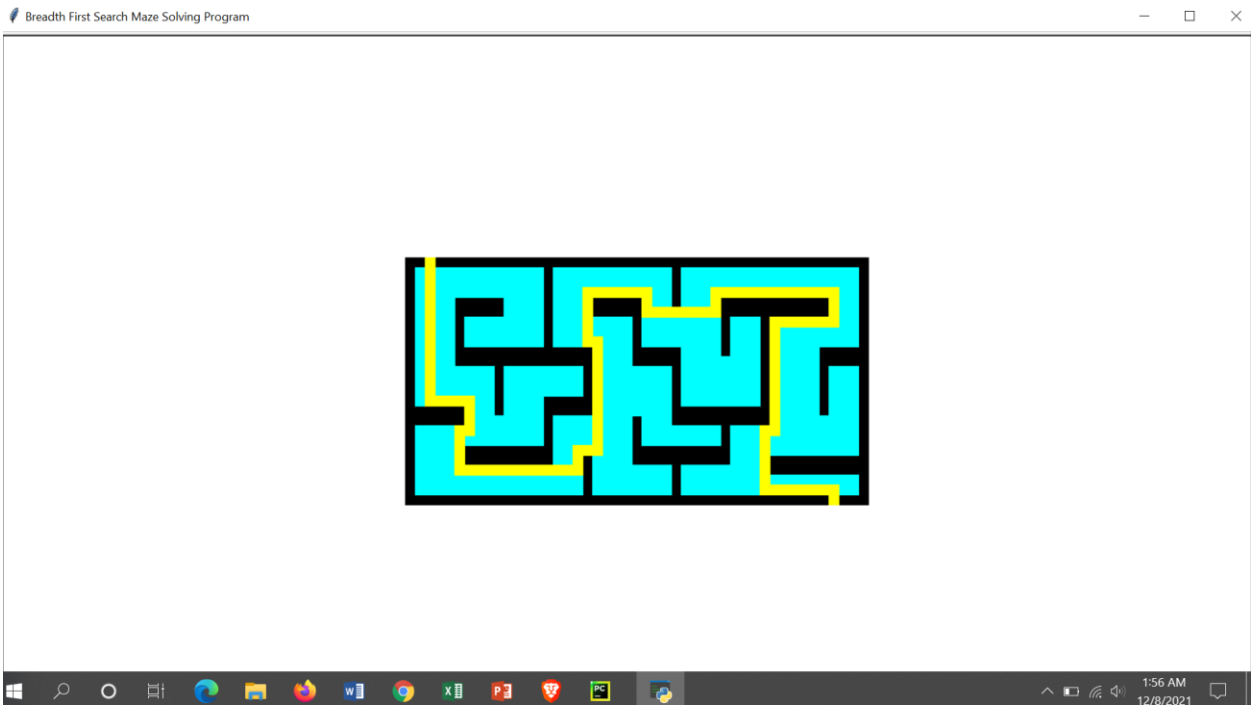
```
        turtle.Turtle.__init__(self)
        self.shape("square")
        self.color("cyan")
        self.shapesize(0.5)
        self.penup()
        self.speed(0)

class Blue(turtle.Turtle):
    def __init__(self):
        turtle.Turtle.__init__(self)
        self.shape("square")
        self.color("blue")
        self.shapesize(0.5)
        self.penup()
        self.speed(0)


# this is the class for the yellow or turtle
class Red(turtle.Turtle):
    def __init__(self):
        turtle.Turtle.__init__(self)
        self.shape("square")
        self.color("red")
        self.shapesize(0.5)
        self.penup()
        self.speed(0)

class Yellow(turtle.Turtle):
    def __init__(self):
        turtle.Turtle.__init__(self)
        self.shape("square")
        self.color("yellow")
        self.shapesize(0.5)
        self.penup()
        self.speed(0)
grid1 = [
"11s11111111111111111111111111111111111111111111",
"10000000000001000000000001000000000000000001",
"10000000000001000000000001000000000000000001",
"10000000000001000000000001000000000000000001",
"10000111110001000011111000100001111111110001",
"10000111110001000011111000000001111111110001",
"10000100000001000000001000000001000100000001",
"10000100000001000000001000000001000100000001",
"10000100000001000000001000000001000100000001",
"10000111111111111110000111110000100010000011111",
"10000111111111111110000111110000000010000011111",
"10000000010000000010000000010000000010000010001",
"10000000010000000010000000010000000010000010001",
"10000000010000000010000000010000000010000010001",
"10000000010000111110000000010000000010000010001",
"11111100010000111110000000011111111110000010001",
"11111100000000100000000100011111111110000000001",
"10000000000000100000000100000000100000000000001",
"10000000000000100000000100000000100000000000001",
"10000011111111100000000111111111100000000000001",
"10000011111111100010000111111111100001111111111",
```

```
"10000000000000000100000001000000001111111111",
"10000000000000000100000001000000000000000001",
"10000000000000000100000001000000000000000001",
"11111111111111111111111111111111111111111e111",
]
def setup_maze(grid):
    lenx = ((len(grid[0])*10)/2)-10
    lenx -= (lenx*2)
    leny = ((len(grid)*10)/2)-5                              # 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 = lenx + (x * 10)         # move to the x location on
the screen staring at -588
            screen_y = leny - (y * 10)          # move to the y location of
the screen starting at 288

            if character == "1":
                maze.goto(screen_x, screen_y)          # move pen to the x and
y locaion 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 == "0" or character == "e":
                path.append((screen_x, screen_y))      # add " " and e to path
list

            if character == "e":
                cyan.color("purple")
                cyan.goto(screen_x, screen_y)       # send green sprite to
screen location
                end_x, end_y = screen_x,screen_y     # assign end locations
variables to end_x and end_y
                cyan.stamp()
                cyan.color("cyan")

            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 endProgram():
    wn.exitonclick()
    sys.exit()

#BFS ALGORITHM IMPLEMENTATION
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 - 10, y) in path and (x - 10, y) not in visited:  # check the
cell on the left
            cell = (x - 10, y)
            solution[cell] = x, y    # backtracking routine [cell] is the
previous cell. x, y is the current cell
            blue.goto(cell)         # identify frontier cells
            blue.stamp()
            frontier.append(cell)   # add cell to frontier list
            visited.add((x-10, y))  # add cell to visited list

        if (x, y - 10) in path and (x, y - 10) not in visited:  # check the
cell down
            cell = (x, y - 10)
            solution[cell] = x, y
            blue.goto(cell)
            blue.stamp()
            frontier.append(cell)
            visited.add((x, y - 10))
            print(solution)

        if(x + 10, y) in path and (x + 10, y) not in visited:   # check the
cell on the  right
            cell = (x + 10, y)
            solution[cell] = x, y
            blue.goto(cell)
            blue.stamp()
            frontier.append(cell)
            visited.add((x +10, y))

        if(x, y + 10) in path and (x, y + 10) not in visited:  # check the
cell up
            cell = (x, y + 10)
            solution[cell] = x, y
            blue.goto(cell)
            blue.stamp()
            frontier.append(cell)
            visited.add((x, y + 10))
        cyan.goto(x,y)
        cyan.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
```

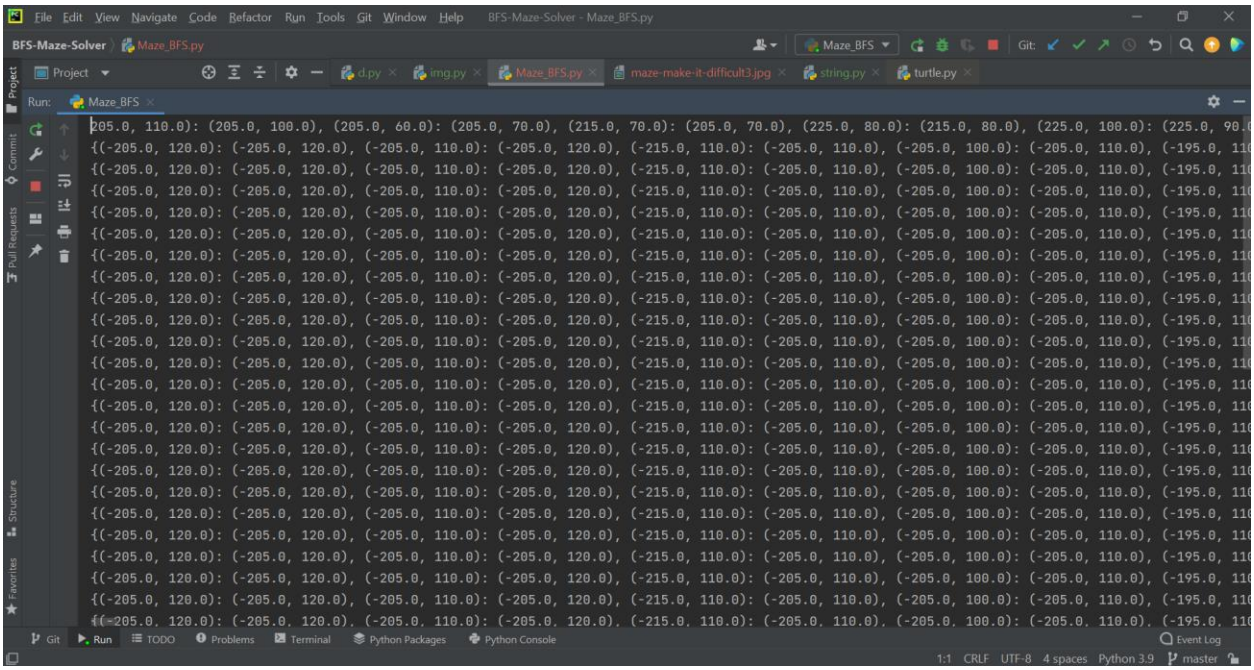
```
key

# set up classes
maze = Maze()
red = Red()
blue = Blue()
cyan = Cyan()
yellow = Yellow()

# setup lists
walls = []
path = []
visited = set()
frontier = deque()
solution = {}                          # solution dictionary


# main program starts here ####
setup_maze(grid1)
search(start_x,start_y)
backRoute(end_x, end_y)
wn.exitonclick()
```

OUTPUT REPRESENTED GRAPHICALLY-



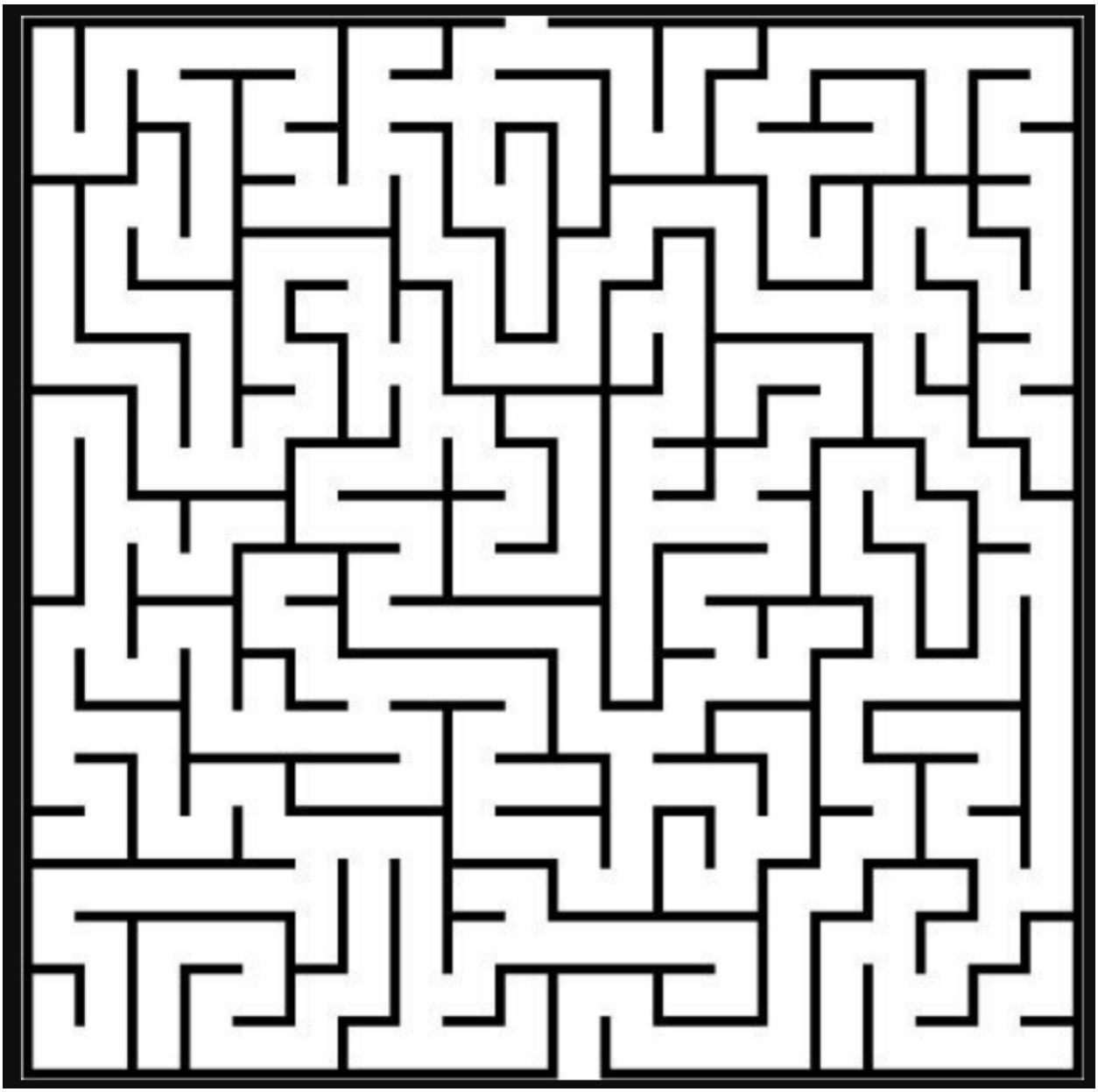
Here in the above graphically represented simple maze file, the algorithm has traversed through the entire maze which was represented in 'CYAN' color and the shortest path is highlighted in yellow color

For the above output of simple maze file, the sample co-ordinate mapping is as follows as it traverses through the each cell of the grid. Co-ordinate space is very large so pasting a small part for reference.


File Edit View Navigate Code Refactor Run Tools Git Window Help
BFS-Maze-Solver - Maze_BFS.py
BFS-Maze-Solver
Maze_BFS.py
Maze_BFS
Git:
Project
d.py
img.py
Maze_BFS.py
maze-make-it-difficult3.jpg
string.py
turtle.py
Run:
Maze_BFS
205.0, 110.0): (205.0, 100.0), (205.0, 60.0): (205.0, 70.0), (215.0, 70.0): (205.0, 70.0), (225.0, 80.0): (215.0, 80.0), (225.0, 100.0): (225.0, 90.0
{(-205.0, 120.0): (-205.0, 120.0), (-205.0, 110.0): (-205.0, 120.0), (-215.0, 110.0): (-205.0, 110.0), (-205.0, 100.0): (-205.0, 110.0), (-195.0, 110
Structure
Favorites
Commit
Pull Requests
Git
Run
TODO
Problems
Terminal
Python Packages
Python Console
Event Log
1:1 CRLF UTF-8 4 spaces Python 3.9 master

CODE AND OUTPUT OF MODERATE MAZE FILE.

INPUT-



PROGRAM CODE-

```
import turtle                    # import turtle library
import time
import sys
from collections import deque

wn = turtle.Screen()               # define the turtle screen
wn.bgcolor("white")                # set the background colour
```

```
wn.title("Breadth First Search Maze Solving Program")
wn.setup(width=1.0,height=1.0)                # setup the dimensions of the
working window


# this is the class for the Maze
class Maze(turtle.Turtle):                 # define a Maze class
    def __init__(self):
        turtle.Turtle.__init__(self)
        self.shape("square")              # the turtle shape
        self.color("black")
        self.shapesize(0.5)               # colour of the turtle
        self.penup()                      # lift up the pen so it do not leave
a trail
        self.speed(0)

# this is the class for the finish line - green square in the maze
class Cyan(turtle.Turtle):
    def __init__(self):
        turtle.Turtle.__init__(self)
        self.shape("square")
        self.color("cyan")
        self.shapesize(0.5)
        self.penup()
        self.speed(0)

class Blue(turtle.Turtle):
    def __init__(self):
        turtle.Turtle.__init__(self)
        self.shape("square")
        self.color("blue")
        self.shapesize(0.5)
        self.penup()
        self.speed(0)


# this is the class for the yellow or turtle
class Red(turtle.Turtle):
    def __init__(self):
        turtle.Turtle.__init__(self)
        self.shape("square")
        self.color("red")
        self.shapesize(0.5)
        self.penup()
        self.speed(0)

class Yellow(turtle.Turtle):
    def __init__(self):
        turtle.Turtle.__init__(self)
        self.shape("square")
        self.color("yellow")
        self.shapesize(0.5)
        self.penup()
        self.speed(0)
grid = [
"1111111111111111111111111111111111111111111111111s1111111111111111111111111111
11111111111111111111111",
```

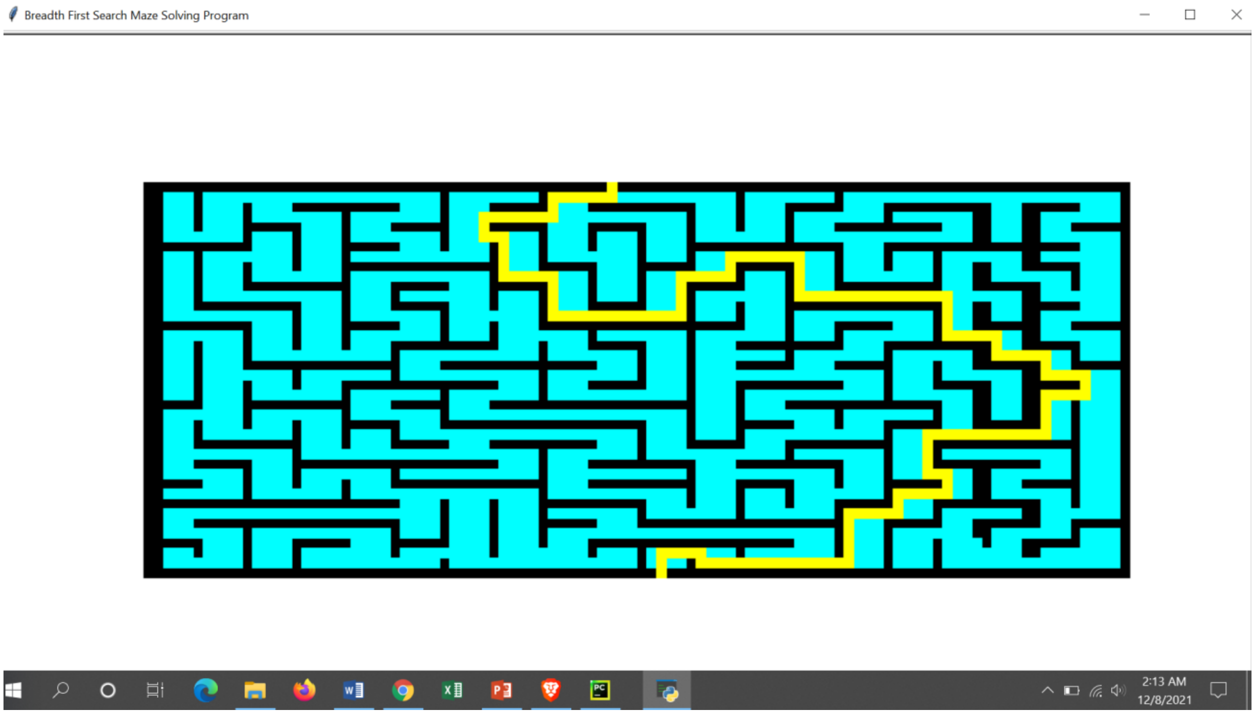
```
"11000100000000000000000000001000000000100000000000000000001000000000100000
000000000000000000000001",
"11000100001000011111111111000010000111111000011111111111000010000111111000001
1111111111000111111100001",
"11000100001000000000100000000010000000000000000000000000100001000010000000001
000000001100011000000001",
"11000100001111110000100001111110000111111000011111100001000010000100001111111
1111100011000110001111111",
"11000000001000010000100000000010000000001000010000100001000000000100000000000
000000001100011000000001",
"11111111111000010000111111000010000100001000010000100001111111111111111100001
1111111111111111111100001",
"11000100000000010000100000000000000100001000000000100001000000000000000100001
000010000000011000000001",
"11000100001000010000111111111111111000011111100001111110000111111000010000100001
000010001100011111100001",
"11000100001000000000100000000000000100000000010000100000000010000100001000000
000010001100000000100001",
"11000100001111111111100001111110000111111000010000100001111110000100001111111
1111100011111111000100001",
"11000100000000000000100001000000000100001000010000100001000000000100000000000
000000000000011000000001",
"11000111111111111100001000011111100001000010000111111000010000100001111111111
1111100011000111111100001",
"11000000000000010000100000000010000000001000000000000000100001000010000000000
000010001100011000000001",
"11111111111000010000111111000010000100001111111111111111111100001000011111
000010001111111000111111",
"11000000001000010000100000000010000100000000010000000001000000000100001000000
000010000000011000000001",
"11000100001000010000100001111111111100001000011111100001000011111111111100001
1111111111000111111100001",
"11000100001000000000000001000000000000000100000000010000100000000010000000001
000000001100000000100001",
"11000100001111111111111110000111111111111111000010000100001111110000111111
000010001111111000111111",
"11000100000000010000000001000000000000000100000000010000100000000000000000001
000010000000011000000001",
"11000100001000010000111111111111111100001000011111100001000011111111111100001
0000111111000111111100001",
"11000100001000000000100000000010000000001000000000000000100001000000000000001
000000001100011000000001",
"11111100001111111111100001111110000111111111111111111111100001000011111111111
1111100011000110001000001",
"11000000001000000000100000000010000000000000000000000000100001000000000100000
000010001100011000100001",
"11000100001000010000111111000011111111111111111111110000100001111110000100001
1111100011111111000100001",
"11000100000000010000100001000000000000000000000000000100001000010000000000000001
000000000000000000100001",
"11000111111111111000010000111111000011111111111100001000011111100001111111111
0000111111111111111100001",
"11000000000000010000000000000000000000000100000000010000000000000001000000001
000010000000000000100001",
"11000111111000011111111111111111111111000010000111111111111000011111111111100001
0000111111111111000100001",
"11000000001000010000000000100000000000000100000000000000100000000000000100001
```

```
0000000011000000000100001",
"1111110000100001000010000111111111111111000011111111111000011111100001000011111100001111100001",
"1100000000100000000010000000000000000000100000000000001000010000100000000010000000000100001",
"1111111111111111111111111100001000010000111111111110000100001000010000111111000011111111100001",
"1100000000000000000000000000010000100001000000000100000000010000000001000000000100001",
"1100011111111111111111111111000010000100001111110000111111111111111111111000011111100001111111",
"1100000000100000000000000100001000010000100000000000000000000000000000100001000000000100001",
"1111110000100001111110000111111000010000100001111111111111111111111000010000100001111100001",
"1100010000100001000000000100000000010000000001000010000000001000000000100001000000000100001",
"1100000000100001000000000000000100000000000000000001000010000000000000000000100001000000000001",
"1111111111111111111111111111111111111111111111111111e1111111111111111111111111111111111111111",
]
def setup_maze(grid):
    lenx = ((len(grid[0])*10)/2)-10
    lenx -= (lenx*2)
    leny = ((len(grid)*10)/2)-5                          # 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 = lenx + (x * 10)         # move to the x location on
the screen staring at -588
            screen_y = leny - (y * 10)          # move to the y location of
the screen starting at 288

            if character == "1":
                maze.goto(screen_x, screen_y)         # move pen to the x and
y locaion 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 == "0" or character == "e":
                path.append((screen_x, screen_y))     # add " " and e to path
list

            if character == "e":
                cyan.color("purple")
                cyan.goto(screen_x, screen_y)       # send green sprite to
screen location
                end_x, end_y = screen_x,screen_y     # assign end locations
variables to end_x and end_y
```

```
                cyan.stamp()
                cyan.color("cyan")

            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 endProgram():
    wn.exitonclick()
    sys.exit()

#BFS ALGORITHM IMPLEMENTATION
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 - 10, y) in path and (x - 10, y) not in visited:  # check the
cell on the left
            cell = (x - 10, y)
            solution[cell] = x, y    # backtracking routine [cell] is the
previous cell. x, y is the current cell
            blue.goto(cell)         # identify frontier cells
            blue.stamp()
            frontier.append(cell)   # add cell to frontier list
            visited.add((x-10, y))  # add cell to visited list

        if (x, y - 10) in path and (x, y - 10) not in visited:  # check the
cell down
            cell = (x, y - 10)
            solution[cell] = x, y
            blue.goto(cell)
            blue.stamp()
            frontier.append(cell)
            visited.add((x, y - 10))
            print(solution)

        if(x + 10, y) in path and (x + 10, y) not in visited:   # check the
cell on the  right
            cell = (x + 10, y)
            solution[cell] = x, y
            blue.goto(cell)
            blue.stamp()
            frontier.append(cell)
            visited.add((x +10, y))

        if(x, y + 10) in path and (x, y + 10) not in visited:  # check the
cell up
            cell = (x, y + 10)
            solution[cell] = x, y
```

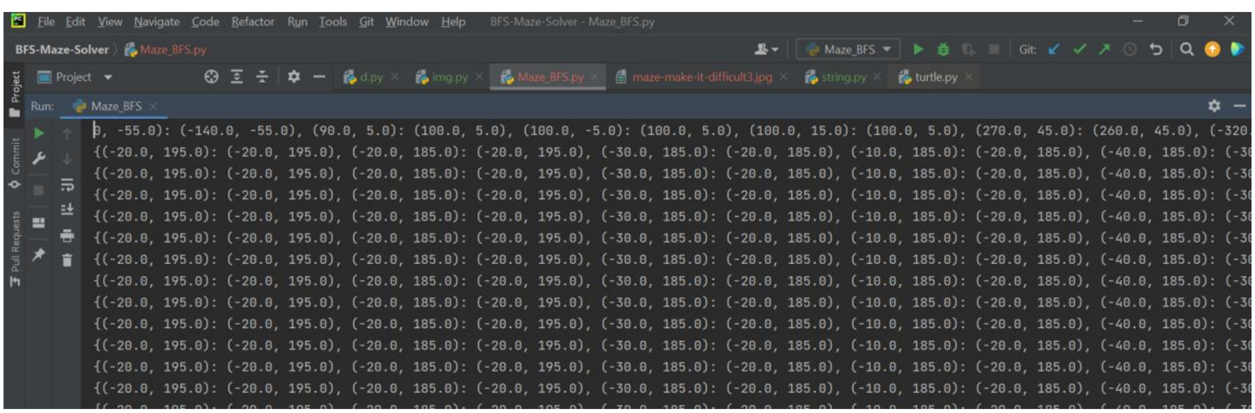
```
            blue.goto(cell)
            blue.stamp()
            frontier.append(cell)
            visited.add((x, y + 10))
        cyan.goto(x,y)
        cyan.stamp()


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

# set up classes
maze = Maze()
red = Red()
blue = Blue()
cyan = Cyan()
yellow = Yellow()

# setup lists
walls = []
path = []
visited = set()
frontier = deque()
solution = {}                           # solution dictionary


# main program starts here ####
setup_maze(grid)
search(start_x,start_y)
backRoute(end_x, end_y)
wn.exitonclick()
```

OUTPUT FOR MODERATE MAZE FILE IS AS FOLLOWS-



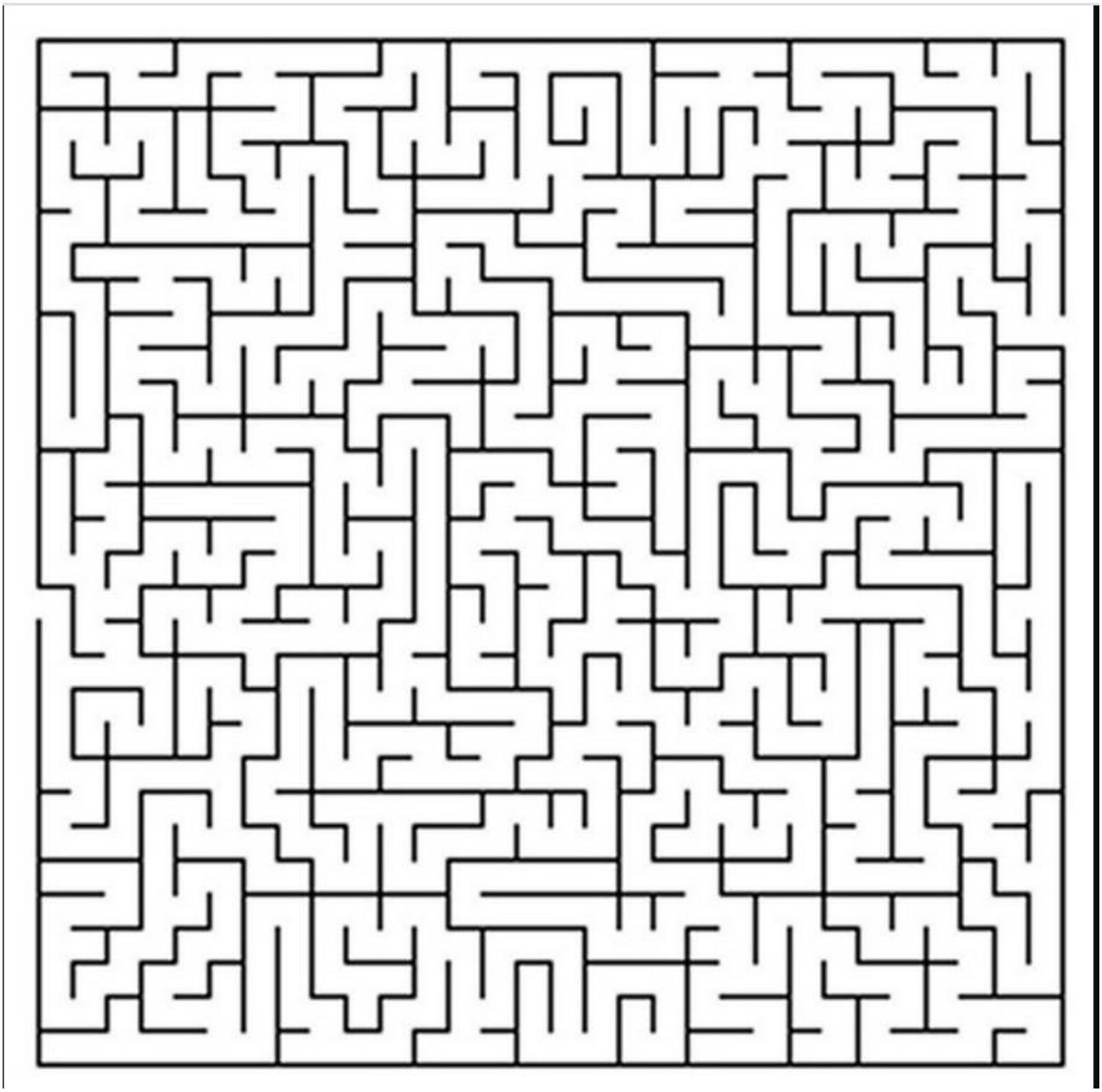
Here in the above graphically represented moderate maze file, the algorithm has traversed through the entire maze which was represented in 'CYAN' color and the shortest path is highlighted in yellow color

For the above output of simple maze file, the sample co-ordinate mapping is as follows as it traverses through the each cell of the grid. Co-ordinate space is very large so pasting a small part for reference.



- **CODE AND OUTPUT OF DIFFICULT MAZE FILE.**

INPUT OF DIFFICULT MAZE FILE-



PROGRAM CODE-

```
import turtle                    # import turtle library
import time
import sys
from collections import deque
```

```
wn = turtle.Screen()                # define the turtle screen
wn.bgcolor("black")                 # set the background colour
wn.title("A BFS Maze Solving Program")
wn.setup(width=1.0,height=1.0)                 # setup the dimensions of the
working window


# this is the class for the Maze
class Maze(turtle.Turtle):                # define a Maze class
    def __init__(self):
        turtle.Turtle.__init__(self)
        self.shape("square")              # the turtle shape
        self.color("white")
        self.shapesize(0.5)               # colour of the turtle
        self.penup()                      # lift up the pen so it do not leave
a trail
        self.speed(0)

# this is the class for the finish line - green square in the maze
class Green(turtle.Turtle):
    def __init__(self):
        turtle.Turtle.__init__(self)
        self.shape("square")
        self.color("green")
        self.shapesize(0.5)
        self.penup()
        self.speed(0)

class Blue(turtle.Turtle):
    def __init__(self):
        turtle.Turtle.__init__(self)
        self.shape("square")
        self.color("blue")
        self.shapesize(0.5)
        self.penup()
        self.speed(0)


# this is the class for the yellow or turtle
class Red(turtle.Turtle):
    def __init__(self):
        turtle.Turtle.__init__(self)
        self.shape("square")
        self.color("red")
        self.shapesize(0.5)
        self.penup()
        self.speed(0)

class Yellow(turtle.Turtle):
    def __init__(self):
        turtle.Turtle.__init__(self)
        self.shape("square")
        self.color("yellow")
        self.shapesize(0.5)
        self.penup()
        self.speed(0)
grid2 = [
```

```
"1111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111
11111",
"1100000000000000000000110000000000000000000000000000110000000001000000000000
00000000000000000011000000000000000000011000000000000000000100000000011000000
00001",
"1100001111110000111111100011111100000111111111111111100001000010000111111110
00011111111111100011111111110000011111110000111111111110000111111100001100001
00001",
"1100000000010000000000000011000000000000001100000000000000100001000000000110
00010000000001100011000000000000000000011000000000000001000000000000000000001
00001",
"1111111111111111111111111111111111111100001100011111111111000011111111111110
00010000100001100011000110000111111000011111100000100001111111111111111100001
00001",
"1100000000010000000000110001100000000000001100000000110000000001000000000110
00010000100001100011000110000100001000000000000000100001000000000000001100001
00001",
"1100001000010000100001100011000011111111111111111000110000100001000011000110
00011111100001100011000110000100001000011111111111111111000011111100001100001
11111",
"1100001000000000100001100011000000000100000000011000110000100000000011000110
00000000000001100000000110000100000000000000110000100000000100000000011000000
00001",
"1100001111111111100001100011111100001000011000110001111111111111111111000110
00010000111111111111111111111000011111100001100001000011111100001111111111110
00001",
"1100000000010000000001100000000010000000001100011000000000100000000000000000
00010000000000000011000000000000001000000000110000000000000100000000011000000
00001",
"1111110000010000111111111110000011111100001100011111100000111111111111111111
11110000111111000011000011111111110000111111111111111111111111111100001100001
11111",
"1111110000010000111111111110000011111100001100011111100000111000000000000111
11110000111111000011000000000000001000011111111111111111111111111100001100001
11111",
"1100000000010000000000000000000000000000001100000000000000100000000000000110
00000000100000000011000000000000001000000000000000000000000000000000011000000
00001",
"1100000000010000000000000000000000000000001100000000000000100000000000000110
00000000100000000011000000000000001000011000000000000001000000000000011000000
00001",
"1100001111111111111111111111111111111111110001111111111100001111111100011100
11111111100001111111111111111111111000011000110000100001000011111111111100001
00001",
"1100001000000000000000000000000001000000000110000000000000010000000001100000
00000000100000000000000000000000001000011000110000100000000010000000001100001
00001",
"1100001111111111100001111110000100001000011000111111111111000010000111111111
11110000111111111111111111111000010000110001100001111111111110000110001111111
00001",
"1100000000010000000000000011000000000100001100011000000000100001000000000000
00010000000000000000000000000100001000011000110000000000000010000110000000001
00001",
"1111110000111111111111000011111111111111111100011000110000111111111111111110
00011111111111111111111100001000010000111111111111111110000100001111111100001
00001",
```

```
"11000010000100000000000000110000000000000000000110001100000000000000000000011000001000000000110000000011000000000100000000000000010000100001000000000011000000000e",
"11000010000100001111111111110000100001111111111110001111111111110000110001100001000010000111111000011111111111111111",
"11000010000100000000000000110000100001000000000000011000000000000000110001100001000010000111111000011111111111111111",
"11000010000100001111111000110000100001000011000111111111000011111111111111111000010000110001100001111111111111111111",
"11000010000100000000011000000000100000000011000110000000000000000000011000000000010000110001100001111111111111111111",
"11000010000111111000011111111111111111111111111110001111111111110000110001111111111111111111111000011111111111111111",
"11000000000100001000011000000000100000000000000110001100000000010000110000000000010000110001100001111111111111111111",
"11111111111100001000011000110000100001111111000111111100001000011111111111111111111111111111111000011111111111111111",
"11111111111100001000000000110000100001111111000111111100001000011111111111111111111111111111111000011111111111111111",
"11000010000000001000000000110000000000000001100000000110000100001000000000000000000000000000000000011000010000100001",
"11000010000111111111111111111111111111111111100011000110000100001000011111000011111111111111111111011000010000100001",
"11000010000111111111111111111111111111111111100011000110000100001000011111000011111111111111111111011000010000100001",
"11000010000000001000000000110000000000000001100011000000000100001000010000011000011000010000100001",
"11000011111000001111111111111111111111100001100011111111111100001111111000111111111111111111111111011000010000100001",
"11000010000000001000000000110000000000000001100011000000000100001000010000000000000000000000000000011000010000100001",
"11000010000111111000011000110000111110000011000110001100001000010000111111000011111111111111111111011000010000100001",
"11000000000100000000011000000000100000000011000000001100001000010000000000000000000000000000000000011000010000100001",
"11111110000100001111111111111111100001111111111111111100001000011111110001111111111111111111111111011000010000100001",
"s00001000000000011000000000100000000011000000000100000000010000100001100011000000000100000000000000011000010000100001",
```

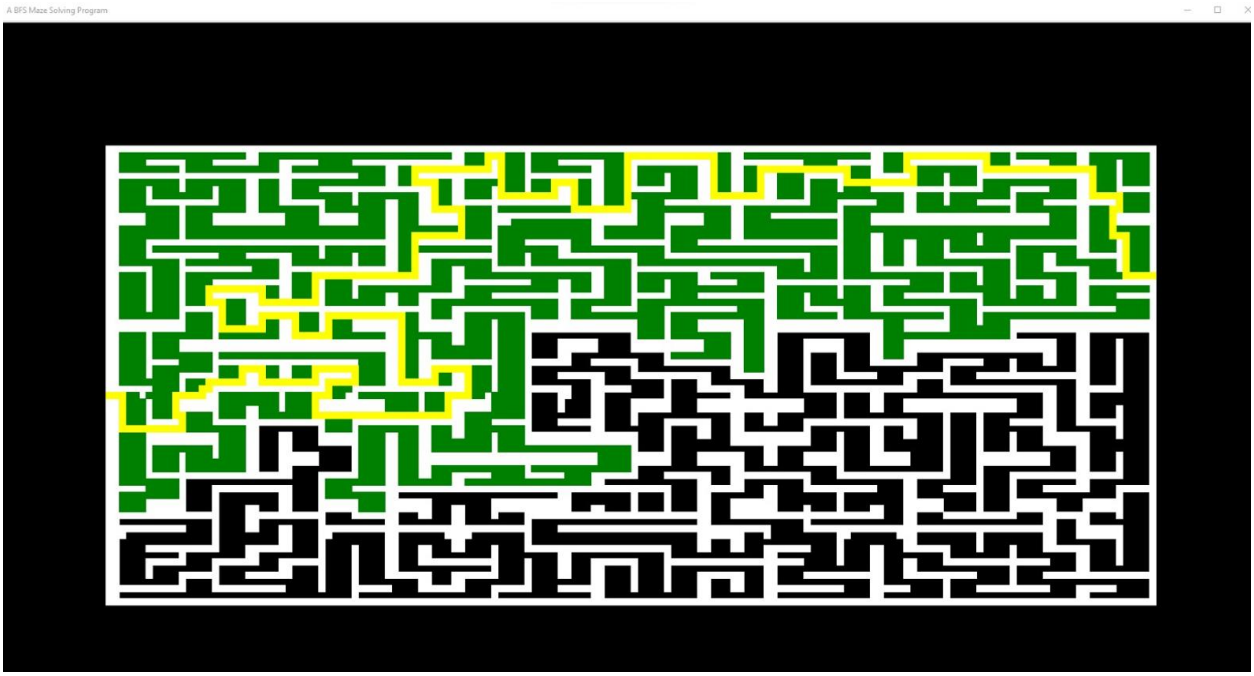
```
"1100001000011111100000000000100000111111111100000100001111110000010000100000110
00011111100001111111111111111100000100001100011111111111111111100001100011000000
00001",
"1100001000011111100001100010000011111111110000010000111111000001000010000110
00011111100001111111111111111100000100001100011111111111111111100001100011000001
00001",
"1100001000000000100001100000000000000000000000000000110000000001000000000110
00010000000000000000110001100000000001000000000000000010000100000000001100011000001
00001",
"11000011111000000111111111111111110000111111111111111110000111111000011111110
00010000111111100011000110000111111111111111110000100001000011111110001111111
00001",
"110000000000000000000110000000000100001000000000011000110000000001000000000110
00000000100001100011000000000010000000001100011000010000100000000011000000001
00001",
"1100001111111111100001100011000011111100001100011000110000100001111111111111
11110000100001100011111111111100001000011000110000100001000010000111111100001
00001",
"1100001111111111100001100011000011111100001100011000000000100001111111111111
11110000100000000001111111111110000100001100000000010000100001000011111100000
00001",
"110000100000000010000110001100000000010000110001100000000010000000000000000000
00010000100000000000000001100000000001000011000000000100001000010000000001100000
00001",
"1100001000010000100001100011111100000100001100011111111111111111111111111111100
0001111110000111111100011000011111100001111110000010000111111111110000110000001
00001",
"110000100001000000000110001111110000010000110001111111111111111111111111111100
0001111110000111111100011000011111100001111110000010000111111111110000110000001
00001",
"110000100001000000000110001100000000001000011000110000000000000001000000000000
00010000000000000000110000000000000000100000000000000000100001000000000000000110001
00001",
"110000111111111111111111111110000111111000011000110001111110000011111100000111
11110000111111100011111111111110000111111111111111111100001000011111111111111111
00001",
"1100000000001000000000000000000000001000000000011000000000110000000000000000000110
000000000000011000110000000000100000000000000001100000000001000010000000001100000
00001",
"111111000001000011111111111111000010000111111111111111111111111111111111111111
1111111110000111111100011000011111111111100001100001111110000100001111111100001
11111",
"1100000000010000100000000001100001000000000011000000000000000000000000000001100000
00010000100001100000000011000000000010000000000110000000001000010000000000000000001
00001",
"110000111111000010000000000110000111111000011111110000000000111111111111000000
00010000100001100011111110000000000100000000011111100000100001111111000011111
00001",
"1100001111110000100001100011000011111100001111111000110000111111111111000110
000100001000011000111111100001000010000110001111110000010000111111100001111111
00001",
"1111111111111111110000111111111111100001111111000110001100001000011111111111111
11111111111111110001111111111111111111111111000110000111111111111100001111111100001
00001",
"110000000000000000100001100000000001000000000011000000000110000000000100000000000000
00000000000000110000000000000000010000000000000000011000000000000000000000011000110000
00001",
```

```
"11111111111100001000011000110000111111111111111111111111111111111000011111111
11111111111111111111111100000111111111111111111111111111111111111110001111111
00001",
"11100000000000001000000000110000110000000011000000011110000000010000000000000
00000000000001111111000100000000011000000011100000000010000000011000111111100
00001",
"11000000000000001000000000110000100000000011000000001100000000010000000000000
00000000000011000110000000000000100000000011000000000100000000011000000000100
00001",
"11000011111111111000011111111000010000100001100011000110000100001111111111111
11111111000011000110001111111000010000110001111111000010000100001111111000010
00001",
"11000000001000000000110000000001000010000110001100000000010000000001100000000
00000001000000000000001100000000010000110000000001000000000100000000011000010
00001",
"11000011111100001111111000111111100001000011000111111111110000100001100011100
11110000111111111111111111111000001000011000110000111111111111111000011000010
00001",
"11000010000000001000000000110000100001000011000000000000001000010000110001100
00010000100000000000000110000000000000011000110000000000010000000000110000000
00001",
"11000010000111111000011111111000010000100001111111000111111100001000011000110
00010000100001111111000110000111111111110001111111111100000100001111111111111
11111",
"11000000000100001000000000000000100001000000000110001100000000010000000001100
00010000100001100011000110000000000000011000000000100000000010000000000000000
00001",
"11111111111100001111111111100000100001111110000111111100001111110000111111100
00010000100001100011000111111111111000011111100000100001111111111000011111100
00001",
"11000000000000000000000000000000000001000000000000000000000100000000000001100
00000000000011000000001100000000000000011000000000100000000000000000011000000
00001",
"11111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111
11111",
]
def setup_maze(grid):
    lenx = ((len(grid[0])*10)/2)-10
    lenx -= (lenx*2)
    leny = ((len(grid)*10)/2)-5                          # 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 = lenx + (x * 10)         # move to the x location on
the screen staring at -588
            screen_y = leny - (y * 10)          # move to the y location of
the screen starting at 288

            if character == "1":
                maze.goto(screen_x, screen_y)         # move pen to the x and
y locaion 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 == "0" or character == "e":
                path.append((screen_x, screen_y))       # add " " and e to path
list

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

            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 endProgram():
    wn.exitonclick()
    sys.exit()

#BFS ALGORITHM IMPLEMENTATION

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 - 10, y) in path and (x - 10, y) not in visited:  # check the
cell on the left
            cell = (x - 10, y)
            solution[cell] = x, y    # backtracking routine [cell] is the
previous cell. x, y is the current cell
            blue.goto(cell)         # identify frontier cells
            blue.stamp()
            frontier.append(cell)    # add cell to frontier list
            visited.add((x-10, y))  # add cell to visited list

        if (x, y - 10) in path and (x, y - 10) not in visited:  # check the
cell down
            cell = (x, y - 10)
            solution[cell] = x, y
            blue.goto(cell)
            blue.stamp()
            frontier.append(cell)
```

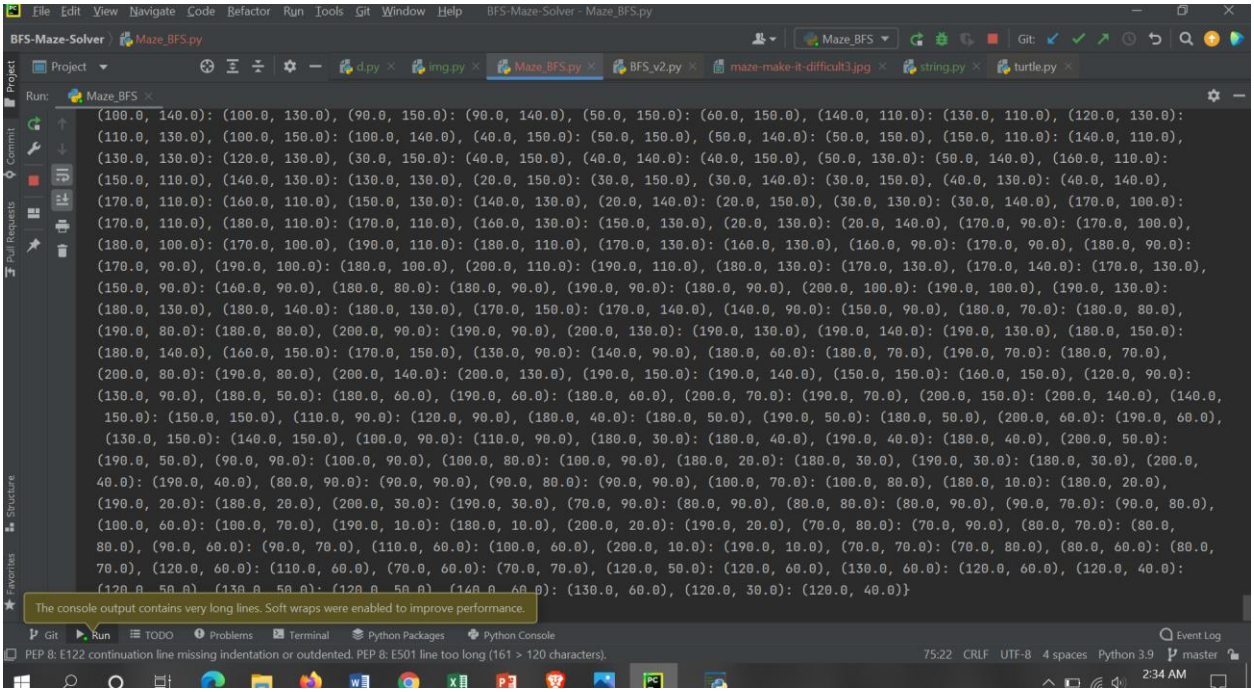
```
            visited.add((x, y - 10))
            print(solution)

        if(x + 10, y) in path and (x + 10, y) not in visited:   # check the
cell on the  right
            cell = (x + 10, y)
            solution[cell] = x, y
            blue.goto(cell)
            blue.stamp()
            frontier.append(cell)
            visited.add((x +10, y))

        if(x, y + 10) in path and (x, y + 10) not in visited:  # check the
cell up
            cell = (x, y + 10)
            solution[cell] = x, y
            blue.goto(cell)
            blue.stamp()
            frontier.append(cell)
            visited.add((x, y + 10))
        green.goto(x,y)
        green.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

# set up classes
maze = Maze()
red = Red()
blue = Blue()
green = Green()
yellow = Yellow()

# setup lists
walls = []
path = []
visited = set()
frontier = deque()
solution = {}                              # solution dictionary


# main program starts here ####
setup_maze(grid2)
search(start_x,start_y)
backRoute(end_x, end_y)
wn.exitonclick()
```

OUTPUT FOR DIFFICULT MAZE FILE IS AS FOLLOWS-



Here in the above graphically represented difficult maze file, the algorithm has traversed through the entire maze which was represented in ‘GREEN’ color and the shortest path is highlighted in yellow color and the remaining paths are not traversed which is in black color.

For the above output of simple maze file, the sample co-ordinate mapping is as follows as it traverses through the each cell of the grid. Co-ordinate space is very large so pasting a small part for reference.



- Below program is the one , which was used to digitize the given maze images.

```
from PIL import Image
import numpy as np
# Open the maze image and make greyscale, and get its dimensions
im = Image.open('maze-make-it-difficult3.jpg').convert('L')
w, h = im.size
# Ensure all black pixels are 0 and all white pixels are 1
binary = im.point(lambda p: p > 128 and 1)

# Resize to half its height and width so we can fit on Stack Overflow,
get new dimensions
binary = binary.resize((w//2,h//2),Image.NEAREST)
w, h = binary.size

# Convert to Numpy array - because that's how images are best stored
and processed in Python
nim = np.array(binary)

# Print that puppy out
for r in range(h):
    for c in range(w):
        print(nim[r,c],end='')
    print()
```

FINAL PROJECT WRITE UP-

1.) What was the problem?

Answer –

I have chosen maze runner project. The problem was, I was given simple, moderate and difficult mazes, which are very convoluted and I have to traverse across the maze to find the shortest path from the start point to the end point of the maze either using BFS, DFS, A* or bi-directional search.

2.) What was the solution you were looking for?

Answer –

I was looking to find the shortest path which is efficient from the start point to the end point marked in the given maze images using breadth first search algorithm from the paths the algorithm has traversed.

3.) How was success of finding that solution to be defined (shortest path, greatest fitness, ...)?

Answer-

The success of finding solution should be defined in the shortest path. Since, the algorithm is not a Kind of genetic algorithm. I have used uninformed search algorithm, so shortest path is yardstick for the measure of performance of the algorithm. The shortest path the algorithm shows after traversing through the maze, the more good it is.

4.) What data was provided, and how was it formatted ?

Answer –

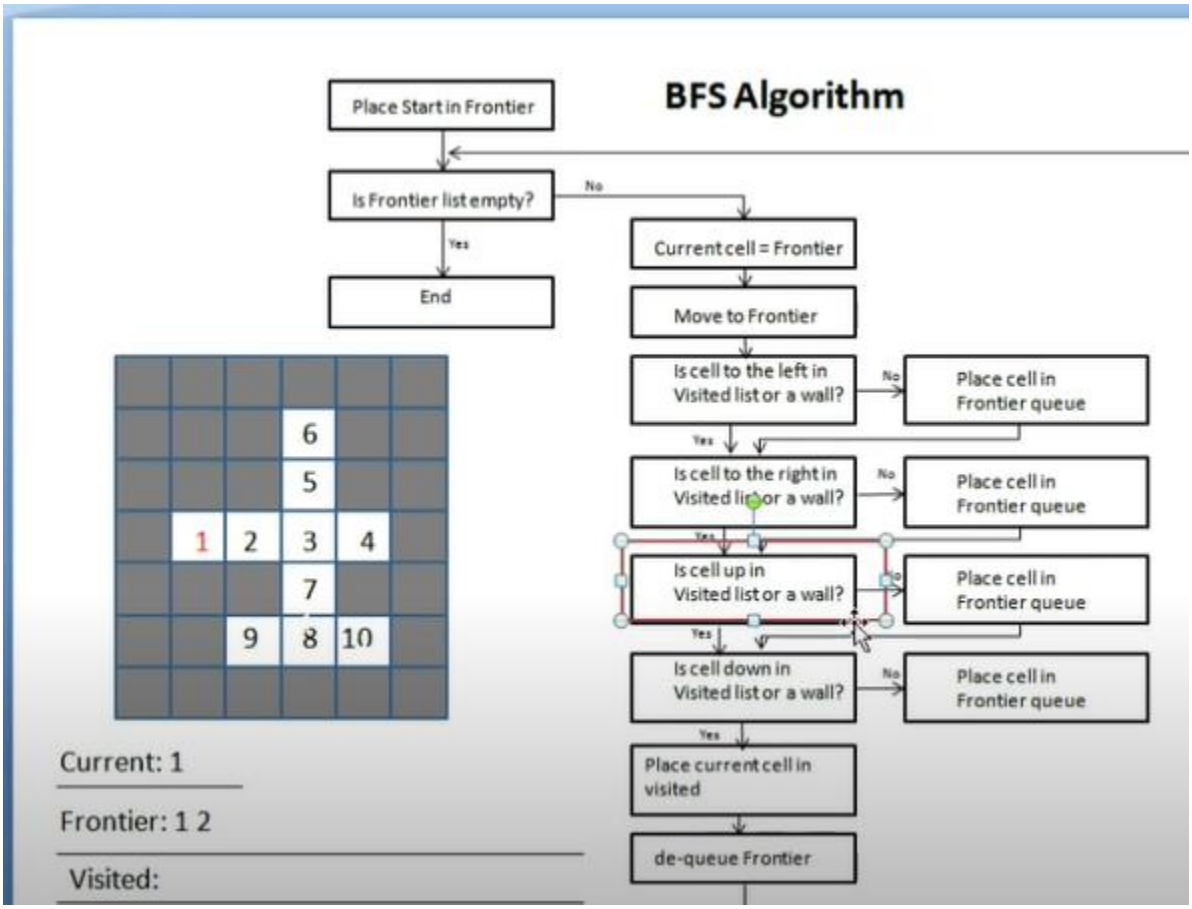
The data provided to solve the problem were images of the mazes. I have digitized the images of the given mazes. I have used '1' to represent maze's walls and spaces with '0' where we can traverse through out the maze. I have used PIL and numpy modules to read the pixels of the image and digitize it. It was formatted in list data structure form of python with each row represented as a string.

5.) How did you approach the problem?

Answer –

Approach was simple, first I have digitized the image as per the maze images and then latter used 'turtle' module which is a great way to create pictures, shapes with a virtual canvas. It is basically a graphical library which helps us to set up the maze and display the solution graphically. I have used turtle methods and functions to represent maze as it is in the given images. I have framed the digitized content of the maze image into a grid system, which is made up of '0's'(indicates space to travel along the maze) and '1's'(to represent walls) when program runs through the grid system. The starting point is denoted by 's' and ending point of the maze by 'e' and then I have used breadth first search algorithm to traverse through the paths of the mazes after exhausting all the paths breadth wise simultaneously, I have used backtracking to find the shortest path using the most efficient route from the paths it has traversed earlier.

BFS IMPLEMENTATION USING MAZE SIMULATION



6.) Why did you select the algorithm that you did? If one was recommended, why is that one a good fit for the problem? What others could have been used?

ANSWER –

- I have selected the breadth first search algorithm, because it is the effective algorithm compared to DFS and I'm very much familiar with BFS algorithm when compared to A* algorithm implementation. BFS is effective because it searches all the paths at the same time (breadth wise) unlike the depth first search algorithm, which only searches the one path at a time (depth wise). In the BFS algorithm implementation, it starts traversing through maze from starting point after the maze set up is done and moves right, up, and down, so every branch it binds and creates another path and does another search, which makes it an efficient algorithm.
- Both BFS and DFS have similar running time, but either may greatly outperform the other on any given problem due to the order in which the cells are visited.

- In terms of space usage, BFS will use more memory compared to DFS and A*,as BFS has to keep multiple paths at the same time in the memory, where as DFS only needs to keep track of a single path at any given time.
- A naive DFS can go into an infinite loop on certain open mazes, whereas on a closed maze it will always finish. I don't think BFS or A* can fall into that trap. ("naive DFS" means one that doesn't mark nodes as "visited" as it traverses them.)
- A* can also be quite efficient compared to naive dfs and bfs. But we need to find a good function to evaluate the cost from your current position to the target.
- Since, we need the shortest path BFS or A* could be the best possible options, since DFS can't give shortest path since it has only one path at any given cell of the grid or maze.
- Since, BFS can give the shortest path and we are asked to find the shortest path, it is a good fit for the problem

7.) How did your attempt work out? What challenges did you encounter?

Answer -

My attempt was pretty good. I believe, I was able to find the shortest path from the start point to end point in the three given mazes using BFS algorithm. Coming to the challenges, I had hard time digitizing the given maze images correctly. I could do it, but when I was comparing it with the images of the mazes, they were few mistakes in the digitized image. 'Stackoverflow' website came to my rescue after fews days of struggle. I did not find BFS algorithm implementation that much hard once I got the digitized image correct.