MINI-PROJECT

Aim:-To develop an AI program that finds and visualizes the shortest path in a maze using Breadth-First Search (BFS) algorithm.

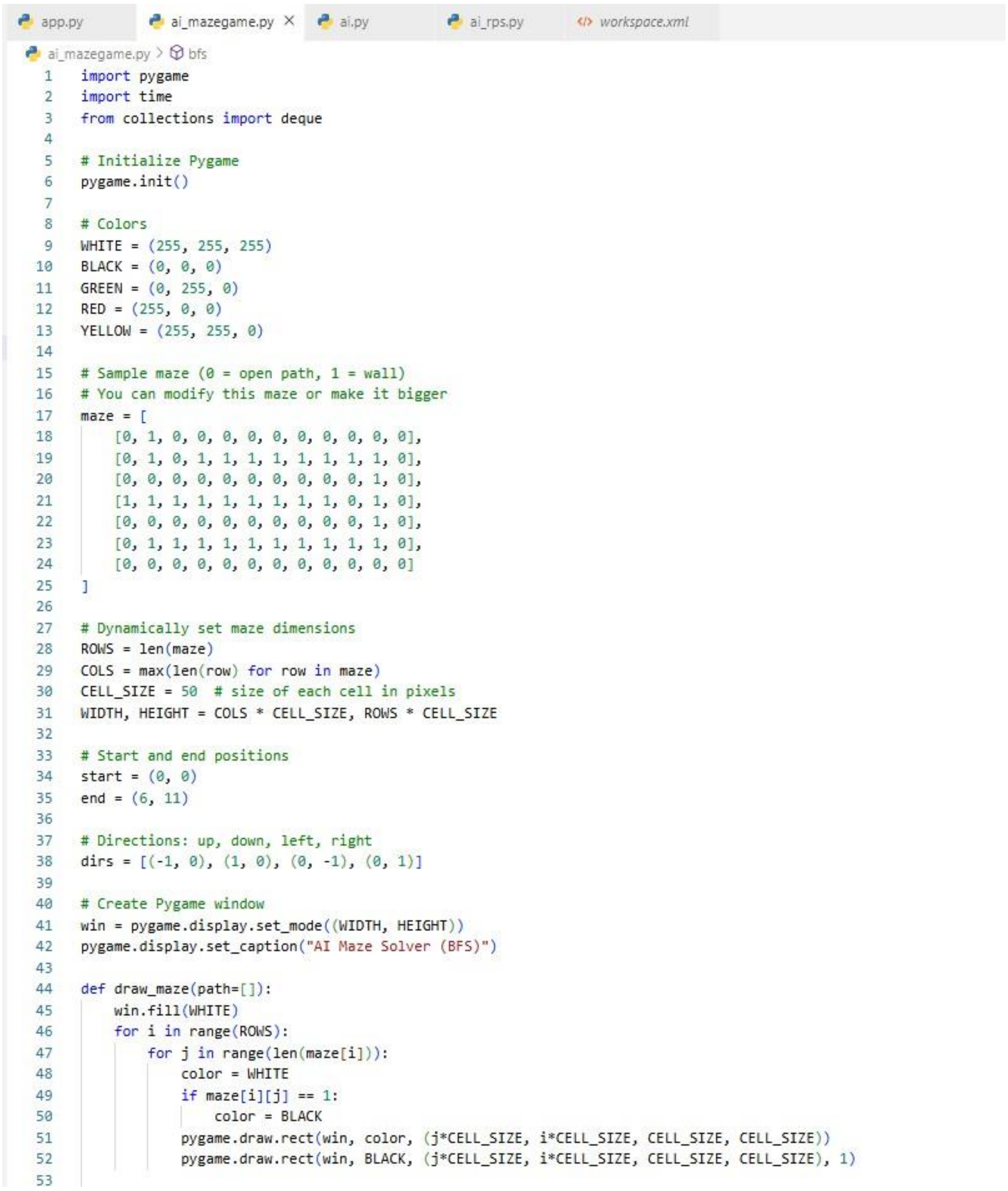
Theory:-

- A **maze** can be represented as a 2D grid of cells: 0 = open path, 1 = wall.
- **BFS (Breadth-First Search)** explores all neighboring nodes level by level and guarantees the **shortest path** in an unweighted maze.
- Visualization tools like **Pygame** can display the maze, start/end points, and the AI's exploration in real-time.
- Each step:
    1. Visit the current node.
    2. Add its valid neighbors to a queue.
    3. Repeat until the end is reached.
    4. Reconstruct the path using parent references.

Algorithm: BFS Maze Solver

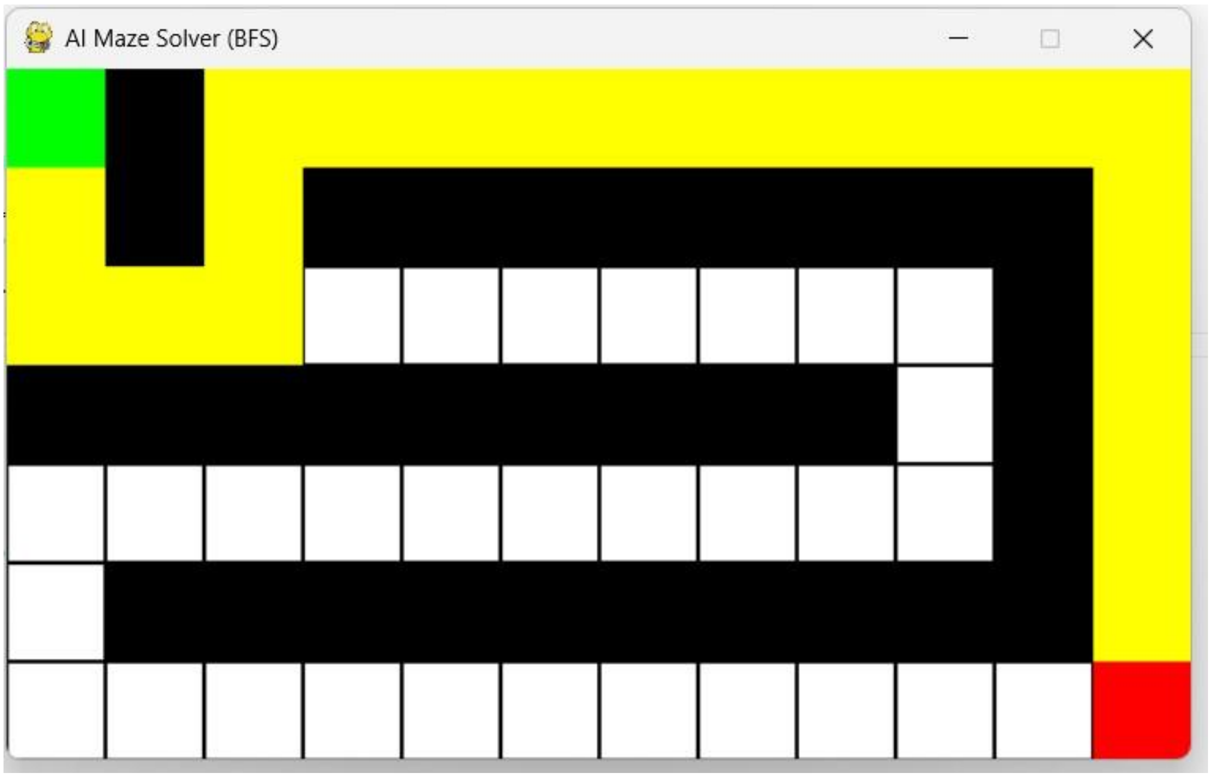
1. **Start**
    - **Mark the start cell as visited and add it to a queue.**
    - **Initialize a parent map to track paths.**
2. **Explore Maze**
    - **While the queue is not empty:**
      a. Remove the front cell from the queue (current cell).
      b. If the current cell is the end, stop.
      c. For each neighbor (up, down, left, right) of the current cell:
        - **Check if it is within maze bounds, not a wall, and not visited.**
        - **Mark it as visited, add it to the queue, and record its parent as the current cell.**
          d. Update the maze visualization (optional step for animation).
3. **Reconstruct Path**
    - **Start from the end cell and trace back to the start using the parent map.**
    - **Mark this as the shortest path.**
4. **Display Result**
    - **Show the maze with start, end, explored cells, and the shortest path highlighted.**

Program:-



```python
import pygame
import time
from collections import deque

# Initialize Pygame
pygame.init()

# Colors
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
GREEN = (0, 255, 0)
RED = (255, 0, 0)
YELLOW = (255, 255, 0)

# Sample maze (0 = open path, 1 = wall)
# You can modify this maze or make it bigger
maze = [
    [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
    [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
]

# Dynamically set maze dimensions
ROWS = len(maze)
COLS = max(len(row) for row in maze)
CELL_SIZE = 50  # size of each cell in pixels
WIDTH, HEIGHT = COLS * CELL_SIZE, ROWS * CELL_SIZE

# Start and end positions
start = (0, 0)
end = (6, 11)

# Directions: up, down, left, right
dirs = [(-1, 0), (1, 0), (0, -1), (0, 1)]

# Create Pygame window
win = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("AI Maze Solver (BFS)")

def draw_maze(path=[]):
    win.fill(WHITE)
    for i in range(ROWS):
        for j in range(len(maze[i])):
            color = WHITE
            if maze[i][j] == 1:
                color = BLACK
            pygame.draw.rect(win, color, (j*CELL_SIZE, i*CELL_SIZE, CELL_SIZE, CELL_SIZE))
            pygame.draw.rect(win, BLACK, (j*CELL_SIZE, i*CELL_SIZE, CELL_SIZE, CELL_SIZE), 1)


    # Draw start and end
    pygame.draw.rect(win, GREEN, (start[1]*CELL_SIZE, start[0]*CELL_SIZE, CELL_SIZE, CELL_SIZE))
    pygame.draw.rect(win, RED, (end[1]*CELL_SIZE, end[0]*CELL_SIZE, CELL_SIZE, CELL_SIZE))

    pygame.display.update()

def bfs(start, end):
    queue = deque([start])
    visited = set([start])
    parent = {}

    while queue:
        current = queue.popleft()

        if current == end:
            break

        for d in dirs:
            ni, nj = current[0] + d[0], current[1] + d[1]

            if 0 <= ni < ROWS and 0 <= nj < len(maze[ni]) and maze[ni][nj] == 0 and (ni, nj) not in visited:
                queue.append((ni, nj))
                visited.add((ni, nj))
                parent[(ni, nj)] = current

        draw_maze(queue)
        time.sleep(0.05)  # visualize step by step

    # Reconstruct path
    path = []
    node = end
    while node != start:
        path.append(node)
        node = parent.get(node, start)
    path.append(start)
    path.reverse()
    return path

def main():
    run = True
    path = bfs(start, end)
    while run:
        draw_maze(path)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False
    pygame.quit()

if __name__ == "__main__":
    main()

```

Output:-



Conclusion:- The AI successfully finds the shortest path from start to end in any given maze. BFS ensures optimal pathfinding, and Pygame visualization helps understand how the algorithm explores the maze step by step. This project demonstrates **AI pathfinding, algorithm efficiency, and real-time visualization**.