Intelligent Maze Game with Dynamic Obstacles Using Pygame

Submitted By: Jahnavi  
Project Type: Individual  
Language Used: Python  
Library Used: Pygame

# Table of Contents

1. Introduction

2. Literature Review

3. Methodology

4. Maze Generation – Grid Based Explanation

5. Flowchart and Block Diagrams

6. Code Snippets and Key Functionalities

7. Results and Discussion

8. Challenges Faced

9. Conclusion

10. References

# 1. Introduction

Game-based simulations have become a compelling method to teach programming logic and enhance algorithmic thinking. This project titled "Intelligent Maze Game with Dynamic Obstacles" is developed using Python and the Pygame library. The game offers an interactive experience where players navigate through a dynamically generated maze containing blinking obstacles. It supports both single-player and two-player modes.  
  
The aim of the project is to combine algorithm design, graphical game interface, and collision detection to build an engaging and educational game environment.

# 2. Literature Review

- Maze Generation Algorithms: Depth-First Search (DFS) generates perfect mazes. Alternatives like Prim’s and Kruskal’s offer multiple path options.  
- Game Development Framework: Pygame simplifies event handling, drawing, and sound effects.  
- Dynamic Obstacle Behavior: Blinking obstacles simulate real-time environmental challenges using timers like pygame.time.get\_ticks().

# 3. Methodology

- Maze Generation: DFS-based recursive backtracking is used.  
- Obstacles: Random cells toggle visibility periodically.  
- Modes:  
 - Single-player (1 circle)  
 - Two-player (2 circles with alternate movement)  
- Controls: Keyboard arrow keys; GUI buttons like Start, Pause, Replay.  
- Game Logic: Collision with visible obstacle ends game. Success tracked by reaching end cell.

# 4. Maze Generation – Grid Based Explanation

The maze is represented by a 2D grid. Each cell is either:  
- 1 (path), or  
- 0 (wall).  
  
This is generated step by step using DFS. Starting from (1,1), the algorithm explores random unvisited neighbors and carves out paths, backtracking when stuck.

# 5. Flowchart and Block Diagrams

Flowchart Steps:  
1. Start Game  
2. Generate Maze (DFS)  
3. Place Obstacles  
4. Select Player Mode  
5. Start Timer  
6. Monitor Key Press & Update Movement  
7. Check Obstacle Collision  
8. Check Completion  
9. End Game / Replay Option  
  
Block Diagram:  
[User Input] → [Game Engine] → [Maze Generator] → [Obstacle Manager] → [Collision Detector] → [Score System] → [Display Output]

# 6. Code Snippets and Key Functionalities

Maze Generation Function:  
```python  
def generate\_maze(rows, cols):  
 stack = []  
 visited = []  
 maze = np.zeros((rows, cols), dtype='int')  
 x, y = 1, 1  
 stack.append((x, y))  
 visited.append((x, y))  
 maze[y][x] = 1  
  
 while stack:  
 neighbours = []  
 x, y = stack[-1]  
 for dx, dy in Directions:  
 nx, ny = x + dx \* 2, y + dy \* 2  
 if (nx, ny) not in visited and 0 <= nx < cols and 0 <= ny < rows:  
 neighbours.append((nx, ny, dx, dy))  
 if neighbours:  
 nx, ny, dx, dy = random.choice(neighbours)  
 maze[y + dy][x + dx] = 1  
 maze[ny][nx] = 1  
 visited.append((nx, ny))  
 stack.append((nx, ny))  
 else:  
 stack.pop()  
 return maze  
```  
  
Obstacle Logic:  
```python  
if pygame.time.get\_ticks() - blink\_timer > blink\_interval:  
 toggle\_obstacle\_visibility()  
 blink\_timer = pygame.time.get\_ticks()  
```

# 7. Results and Discussion

The game runs successfully with dynamic maze generation. Blinking obstacles toggle periodically. Single and two-player functionality tested. Interface allows pause, restart, and scoring.

# 8. Challenges Faced

- Synchronizing obstacle blinking with frame rate.  
- Handling collisions only during obstacle visibility.  
- Ensuring maze connectivity without loops.  
- Player circle freeze/resume on obstacle pathing.  
- Designing a responsive and intuitive GUI.

# 9. Conclusion

This project demonstrates a complete pipeline for a Python-based maze game using algorithmic thinking and GUI design. The game balances educational goals with entertainment and real-time logic.

# 10. References

- Pygame Documentation: https://www.pygame.org/docs/  
- Depth-First Search Maze Generation: https://en.wikipedia.org/wiki/Maze\_generation\_algorithm  
- Python Docs: https://docs.python.org/3/  
- Game Design Blogs: Real Python, GeeksforGeeks, freeCodeCamp

# Game Screenshot

