
SNAKE GAME
USING BFS SEARCH ALGORITHM

PROJECT IDEA

The project implements the classic Snake Game with an AI-driven snake that automatically navigates toward food using the Breadth-First Search (BFS) pathfinding algorithm. The goal is to demonstrate how search algorithms can be applied to solve real-time navigation problems in a constrained grid environment. The AI snake uses BFS to find the shortest path to the food while avoiding collisions with itself and the grid boundaries.

ALGORITHM USED

- Breadth-First Search (BFS) – used to find the shortest path from the snake’s head to the food.
  - Explores all neighboring cells at the present depth before moving to the next level.
  - Ensures the shortest path in an unweighted grid.
  - Time Complexity: O(V + E) where V is vertices (grid cells) and E is edges (movements).
  - Space Complexity: O(V) for visited set and queue.
- Grid-based Movement – Snake moves in four directions: Up, Down, Left, Right.
- Collision Detection – Checks for wall and self-collision.
- Dynamic Obstacle Avoidance – The snake’s body is treated as obstacles during pathfinding.

SYSTEM DESIGN OUTLINE

3.1 MODULE BREAKDOWN

1.bfs_search.py – Contains the BFSPathfinder class.

- find_path(): Returns the shortest path from start to target.
- find_safe_direction(): Converts path to movement direction.
- _is_valid_position(): Validates grid boundaries and obstacles.

2.**snake_game.py – Main game engine using Tkinter.**

- SnakeGame class: Manages game state, rendering, and AI logic.
- _ai_move(): Calls BFS pathfinder to decide next move.
- move_game(): Main game loop.
- _draw(): Renders snake, food, and grid.

3.2 FLOW OF CONTROL

1. **Game initializes snake, food, and BFS pathfinder.**
2. **Each frame:**
    - AI computes path to food using BFS.
    - Snake moves one step along the path.
    - If food is eaten, score increases and new food spawns.
    - If collision occurs, game ends.
3. **Loop continues until game over.**

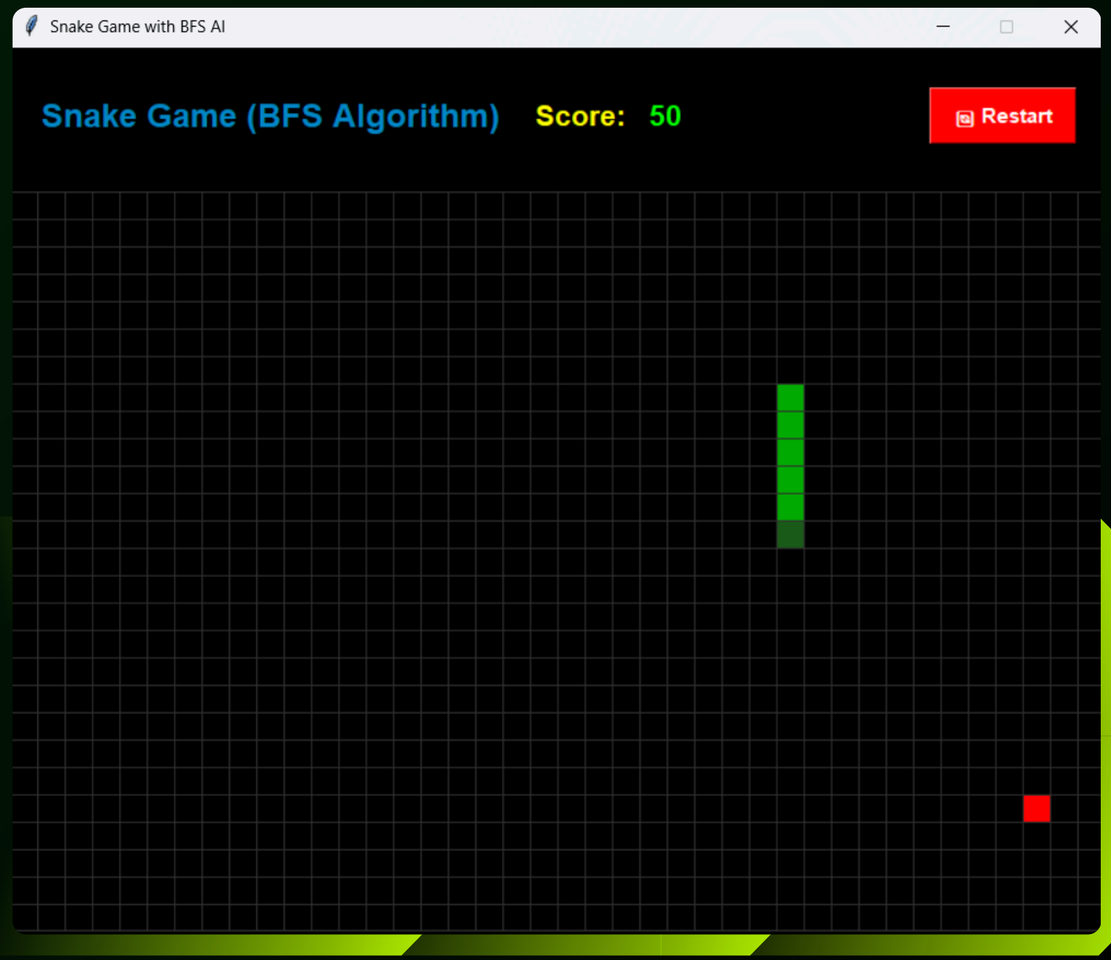
3.3 DATA STRUCTURES

1. **Snake: List of (x, y) tuples.**
2. **Obstacles: Set of snake body positions.**
3. **Queue: deque for BFS traversal.**
4. **Visited Set: Tracks visited cells in BFS.**

HOW TO RUN THE PROJECT

PREREQUISITES

1. Python 3.7+
2. Tkinter (usually included with Python)



STEPS

1. Download both files:
- bfs_search.py
- snake_game.py
2. Place them in the same folder.
3. Run the game:

   python snake_game.py
4. The game window will open automatically.

CONTROLS

1. The game is fully AI-driven — no keyboard input required.
2. Use the "Restart" button to reset the game.

INPUT/OUTPUT SAMPLES

INPUT

1. The game takes no external input from the user.
2. Internally, the AI uses:

- Start: Snake’s head position (x, y)
- Target: Food position (x, y)
- Obstacles: Set of snake body cells

OUTPUT

1. Visual: Grid-based GUI showing:

- Green snake (dark head, lighter body)
- Red food
- Score display

2. Console: No console output; all feedback is in the GUI.
3. Game Over Pop-up: Shows final score when snake collides.

TESTING RESULTS

6.1 FUNCTIONAL TESTING

Test Case	Result
Snake finds food successfully	✅ Pass
Avoids self-collision	✅ Pass
Stops at wall collision	✅ Pass
Path recalculates after eating food	✅ Pass
Restart button works	✅ Pass

6.2 PERFORMANCE TESTING

1. **Grid Size: 40×30 (1200 cells)**
2. **BFS Execution Time: < 10ms per frame**
3. **Smooth Gameplay: 100ms per move (game_speed)**
4. **Memory Usage: Minimal; scales with snake length.**

6.3 EDGE CASES TESTED

1. **Food inside snake body – Regenerated automatically.**
2. **No path to food – Snake continues moving until a path opens.**
3. **Long snake – BFS still finds path if possible.**
4. **Immediate collision on restart – Snake respawns safely.**

6.4 OBSERVATIONS

1. **BFS guarantees the shortest path but can be inefficient for large grids.**
2. **The snake sometimes gets trapped when body blocks all routes.**
3. **Game speeds up with longer snakes due to more obstacles.**

FUTURE IMPROVEMENTS

1. Implement A* algorithm for faster pathfinding.
2. Add difficulty levels (speed, grid size).
3. Include manual play mode for comparison.
4. Add sound effects and visual path highlighting.
5. Implement Hamiltonian cycle for guaranteed win in constrained grids.

CONCLUSION

This project successfully demonstrates the application of BFS in a real-time game environment. It highlights how classic AI search algorithms can be used for navigation and decision-making in dynamic scenarios. The system is modular, extendable, and serves as a practical example of AI integration in game development.

OUR TEAM

Name	ID
Ahmed Ibrahim AboAlmaaty	2301192
Ali Abd Al Hamed Ali	2300424
Mostafa Ahmed Mohamed	2300426
Angluana Amged Salah	2300429
Carol Hosam Zakaria	2301084

THANK YOU