Name	**CHEWI CLINTON SHU**
Matricule	ICTU20241182
Course Lecturer	Dr. Fotsing

ARTIFICIAL INTELLIGENCE CA – MAZE NAVIGATION PROJECT

1. INTRODUCTION AND PROBLEM STATEMENT

This project implements and analyzes a 2D maze navigation game using Python and PyGame. It uses 0 for free paths and 1 for walls. This project creates an interactive maze game that allows both manual navigation (with arrow keys) and automated pathfinding using algorithms like breadth-first search, depth-first search, and generative algorithms. This project makes use of two grids, namely MAZE_1 (10x10) and MAZE_2 (7x8).
The project solves the issue of finding optimal paths in different sectors of our environment, like in game AI.

2. IMPLEMENTATION APPROACH AND DESIGN DECISION

2.1 MAZE REPRESENTATION AND GAME ARCHITECTURE

The maze is implemented using an object-oriented design with 3 primary classes:
***Maze*:**
The maze class uses a grid structure, dimension, start/stop goal position, and makes use of the *is_valid_move()* method ensures boundary checking and wall collisions detection, while *get_neighbors()* returns valid adjacent cells in four directions, that is, right, left, up, and down.

Player:
It manages the player's state, including position, manual navigation with arrow keys, and collision detection, which prevents movement through walls. If the player.png image is missing, it falls back to a drawn sprite.

2.2 SEARCH ALGORITHM IMPLEMENTATION

Depth-first:
It is implemented recursively, exploring as far as possible along each branch before backtracking. The algorithm tracks visited nodes in order and provides a final path if found.

Breadth-First:
It is implemented using a queue (deque) to explore all the neighbors at the present depth before moving to nodes at the next depth level. This guarantees finding the shortest path in terms of the number of cells traversed.

2.3 GENETIC ALGORITHM IMPLEMENTATION

It models the maze as an optimization problem.

Genetic Algorithm Components

Component	Description
Chromosomes Representation	A sequence of integers (0–3) representing movement directions (up=0, down=1, left=2, right=3)
Fitness Function	Evaluates each chromosome based on distance from the goal (Manhattan distance), penalty for hitting a wall, and reward for reaching the goal
Selection	Tournament selection to choose a parent for reproduction
Crossover	Single-point crossover with a probability of 0.8
Mutation	Random gene changes with a probability of 0.15
Elitism	Preserve the best 2 individuals in each generation

Chromosome Length 60

2.4 Visualization

- Walls (black), paths (white), grid lines (gray)
- Start (Green circle), Goal (Red checkered cell)
- Explored nodes is yellow for DFS & BFS or Orange for GA
- Final solution path is purple for DFS & BFS or Orange for GA

2.4.2 Algorithm Visualization

When triggered with the 'D', 'B', or 'G' keys, the respective algorithms show their exploration patterns and final paths:

- DFS shows deep exploration with backtracking
- BFS shows systematic, level-by-level exploration
- GA shows evolving solutions over generations

3. EXPERIMENTAL RESULTS

Multiple executions were done and below are the representative runs.

Maze 1 (10×10, Start: (0,0), Goal: (8,8))

Algorithm Path Length Nodes Time Explored

DFS	17	42	0.000
BFS	17	42	0.000
GA	23	-	0.043 (gen 3)

Maze 2 (8×7, Start: (0,0), Goal: (6,6))

Algorithm Path Length Nodes Time Explored

DFS	11	14	0.000
BFS	11	21	0.000
GA	13	-	0.024 (gen 0)

3.2 ALGORITHM ANALYSIS

3.2.1 DFS

In maze 2 it explored fewer nodes (14) than BFS (21).

STRENGTH:
Low memory required and faster in finding path when the solution is deep in the search tree.

WEAKNESS:
It can get stuck in long non optimal paths.
Do not guarantee the shortest path.

3.2.2 BFS

STRENGTH:
Guarantees the shortest path in terms of numbers of cell.
More predictable than DFS.

3.2.3 GENETIC ALGORITHM IMPLEMENTATION

STRENGTH:
Found solutions very very quickly (generation 0 in MAZE 2, generation 3 in MAZE 1).
It doesn't require full knowledge of the maze.

WEAKNESS:
IT requires more time than BFS AND DFS.

3.3 BEHAVIORAL ANALYSIS

Maze 1:

- ❖ Both BFS and DFS found the optimal path of length 17 after exploring 42 nodes
- ❖ GA found a suboptimal path of length 23 after just 3 generations
- ❖ GA took significantly longer (0.043s) compared to BFS/DFS (0.000s)

Maze 2:

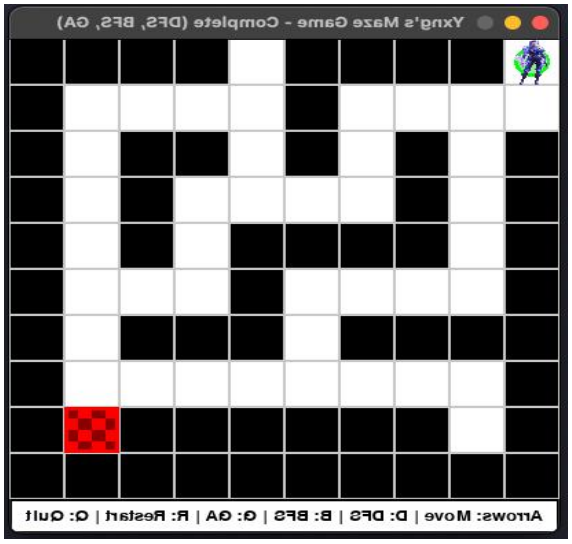
- ❖ Both BFS and DFS found the optimal path of length 11
- ❖ DFS was more efficient than BFS, exploring only 14 nodes vs 21
- ❖ GA found a solution in the initial generation (0) but with a longer path (13)
- ❖ GA was faster in Maze 2 (0.024s) than in Maze 1

GA performs relatively better in Maze 2 (13 vs optimal 11) than in Maze 1 (23 vs 17), as reduced complexity lowers the optimization barrier.

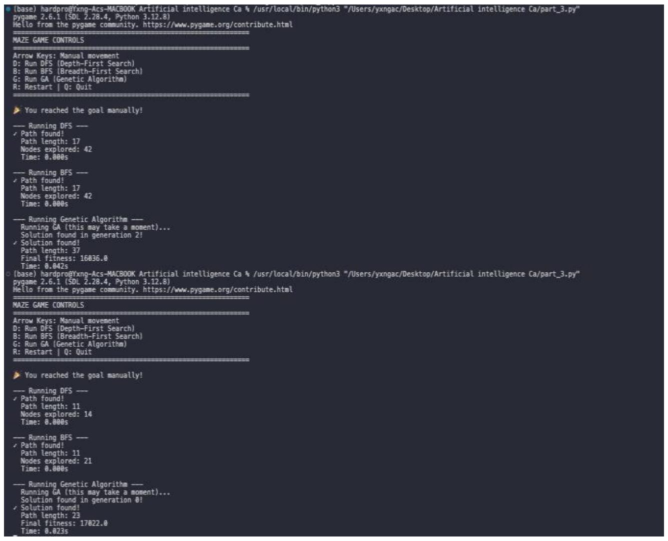
4. LESSONS LEARNED

1. DFS is fast but unpredictable
2. Visualization Importance: Visualizing algorithm behavior greatly enhanced understanding of their exploration patterns and decision-making processes
3. The structure of the maze significantly affected algorithm performance, as seen in the different behaviors between Maze 1 and Maze 2

Screenshots

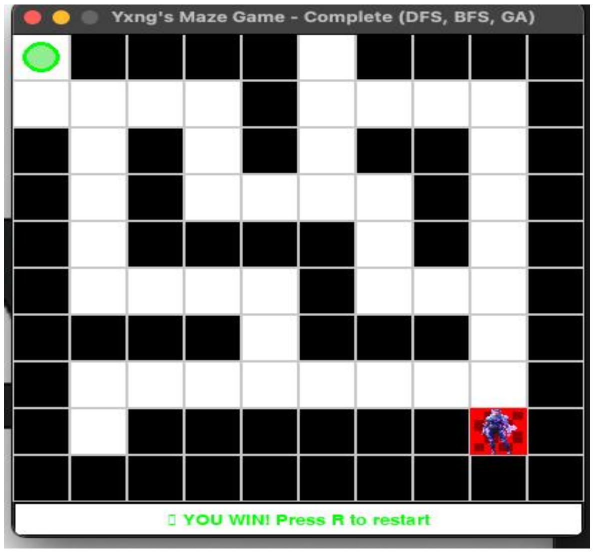

Yxng's Maze Game - Complete (DFS, BFS, GA)
Arrows: Move | D: DFS | B: BFS | G: GA | R: Restart | Q: Quit

MAZE interface

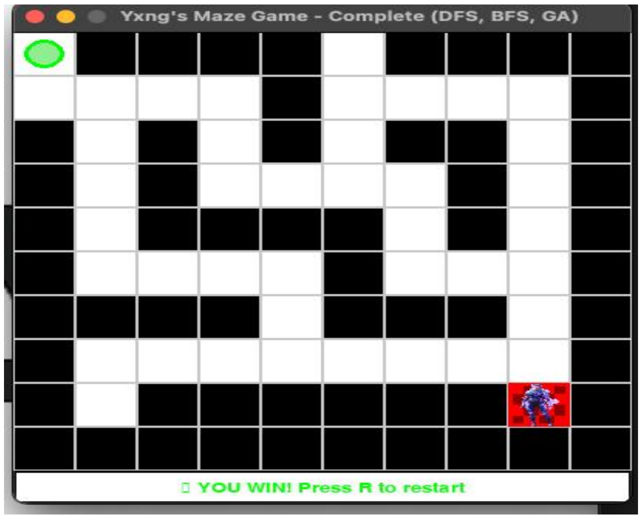


```
(base) hardpro@Yxng-Acs-MACBOOK Artificial intelligence Ca % /usr/local/bin/python3 "/Users/yxngac/Desktop/Artificial intelligence Ca/part_3.py"
pygame 2.6.1 (SDL 2.28.4, Python 3.12.8)
Hello from the pygame community. https://www.pygame.org/contribute.html
============================================================
MAZE GAME CONTROLS
============================================================
Arrow Keys: Manual movement
D: Run DFS (Depth-First Search)
B: Run BFS (Breadth-First Search)
G: Run GA (Genetic Algorithm)
R: Restart | Q: Quit
============================================================

🎉 You reached the goal manually!

--- Running DFS ---
✓ Path found!
  Path length: 17
  Nodes explored: 42
  Time: 0.000s

--- Running BFS ---
✓ Path found!
  Path length: 17
  Nodes explored: 42
  Time: 0.000s

--- Running Genetic Algorithm ---
  Running GA (this may take a moment)...
  Solution found in generation 2!
✓ Solution found!
  Path length: 37
  Final fitness: 16036.0
  Time: 0.042s
(base) hardpro@Yxng-Acs-MACBOOK Artificial intelligence Ca % /usr/local/bin/python3 "/Users/yxngac/Desktop/Artificial intelligence Ca/part_3.py"
pygame 2.6.1 (SDL 2.28.4, Python 3.12.8)
Hello from the pygame community. https://www.pygame.org/contribute.html
============================================================
MAZE GAME CONTROLS
============================================================
Arrow Keys: Manual movement
D: Run DFS (Depth-First Search)
B: Run BFS (Breadth-First Search)
G: Run GA (Genetic Algorithm)
R: Restart | Q: Quit
============================================================

🎉 You reached the goal manually!

--- Running DFS ---
✓ Path found!
  Path length: 11
  Nodes explored: 14
  Time: 0.000s

--- Running BFS ---
✓ Path found!
  Path length: 11
  Nodes explored: 21
  Time: 0.000s

--- Running Genetic Algorithm ---
  Running GA (this may take a moment)...
  Solution found in generation 0!
✓ Solution found!
  Path length: 23
  Final fitness: 17022.0
  Time: 0.023s
```

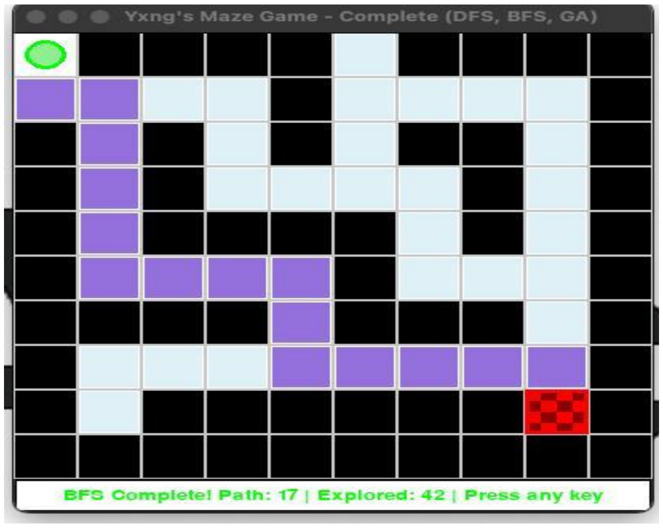
Result of MAZE 1 & MAZE 2, respectively


Yxng's Maze Game - Complete (DFS, BFS, GA)
□ YOU WIN! Press R to restart

DFS Visualization


Yxng's Maze Game - Complete (DFS, BFS, GA)
□ YOU WIN! Press R to restart

GA Visualization



BFS Complete! Path: 17 | Explored: 42 | Press any key

BFS Visualization

5. CONCLUSION

This project successfully implemented a maze navigation game with three different pathfinding approaches: DFS, BFS, and a Genetic Algorithm.

- BFS and DFS consistently found optimal paths in both mazes, with DFS showing better efficiency in the more constrained Maze 2

- GA found solutions very quickly (especially in Maze 2) but with longer paths, highlighting the trade-off between speed and optimality

- The structure of the maze significantly impacted algorithm performance, with Maze 2 generally resulting in faster execution and different efficiency patterns