Artificial Intelligence Assessment 1

Maze Navigation and Search Algorithms Documrntation

Name: Anya Awazi Akuru
Matricule: ICTU20223055
Email: anya.awaziakuru@ictuniversity.edu.cm

Introduction

This project implements a 2D maze navigation game in Python using Pygame and compares three algorithmic approaches to maze solving: Breadth-First Search (BFS), Depth-First Search (DFS), and a Genetic Algorithm (GA). The application visualizes interactive gameplay, animated search progress, and the final solution paths. Objectives include implementing visualization, comparing algorithm performance, and examining how algorithm behavior changes across maze variants.

Problem Description & Representation

The maze is modeled as a 2D integer matrix where 0 = free cell and 1 = wall. The game draws the grid, enforces collision detection for manual player movement, and highlights the start (green) and goal (red checkered). The start is fixed at the top-left corner; the goal is at the bottom-right corner in the provided fixed maze. The implementation supports arbitrary start/goal placements within the maze bounds.

Implementation Approach and Key Design Decisions

- **Language & Tools:** Python 3 + Pygame for rendering and input. Standard library (collections, random) for algorithms.

- **Search Algorithms:** Implemented BFS (queue-based) and DFS (stack-based) as generators to allow step-through animation of frontier and visited nodes.

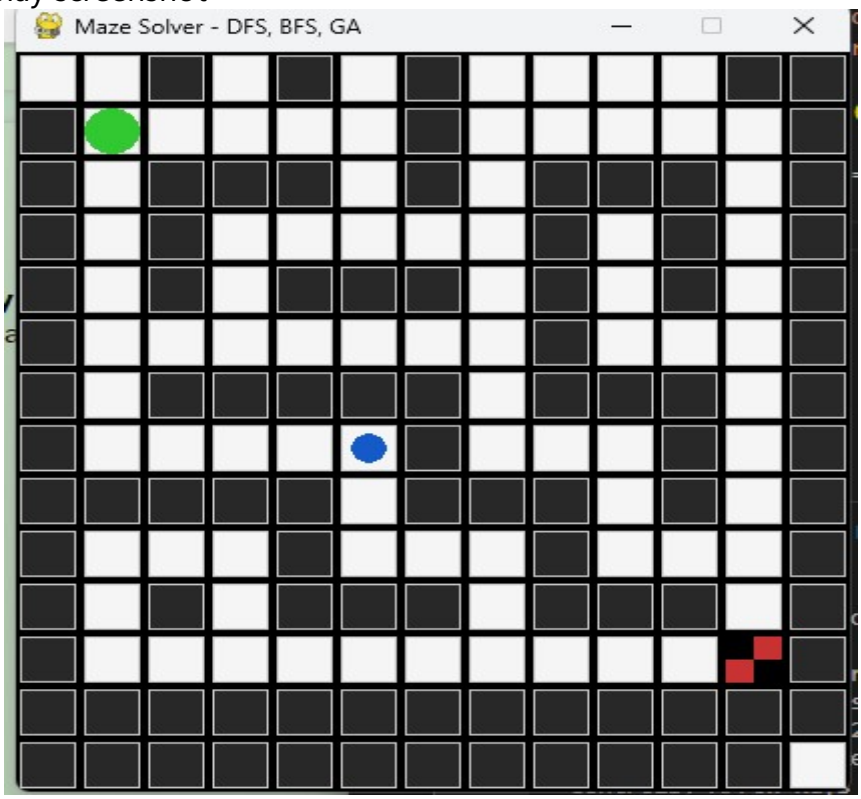
- **GA Design:** Individuals are fixed-length move sequences (U/D/L/R). Fitness = Manhattan distance to goal × weight + collisions × large penalty + small length penalty. GA operators: tournament selection, single-point crossover, mutation.

- **Visualization:** Grid cells colored to show visited nodes, frontier, and final path. Player movement with arrow keys; keys d/b trigger DFS/BFS; g runs GA.

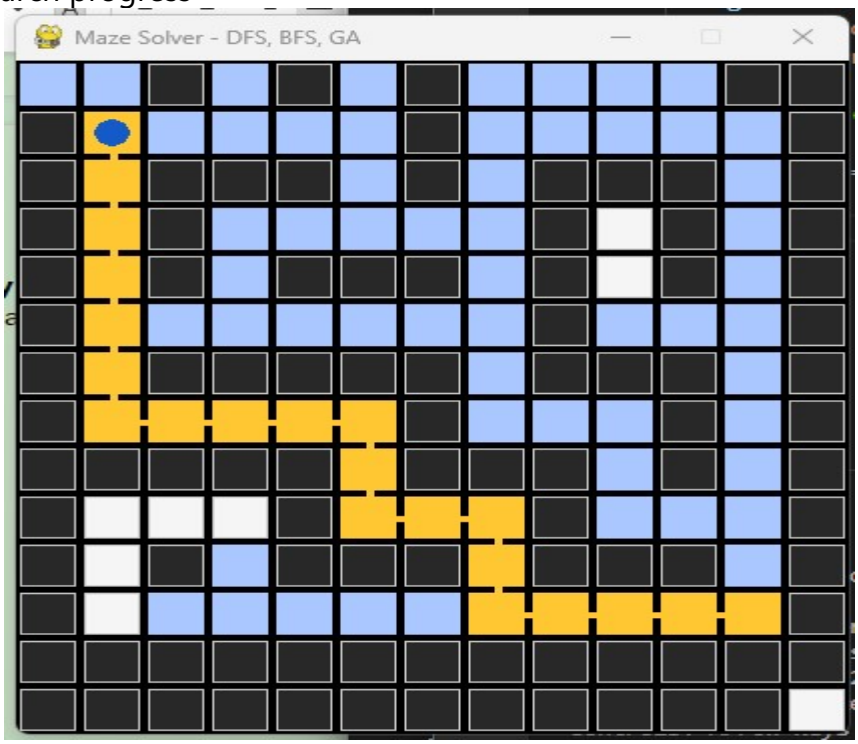
- **Complexity control:** Fixed, moderately sized maze (16×16) balancing visual clarity and meaningful algorithmic performance.

Screenshots

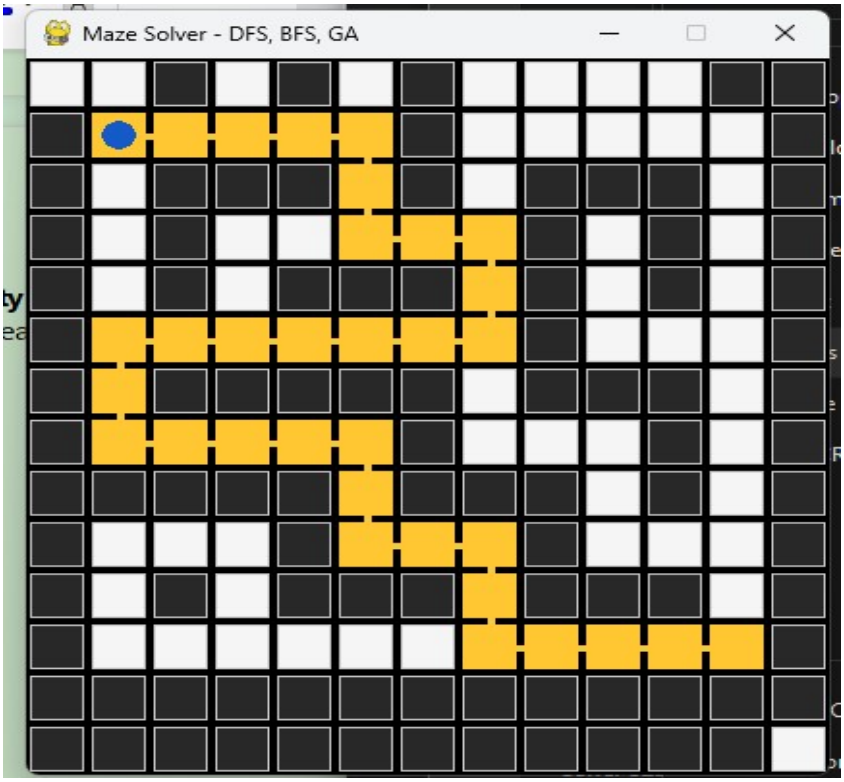
1. Gameplay screenshot



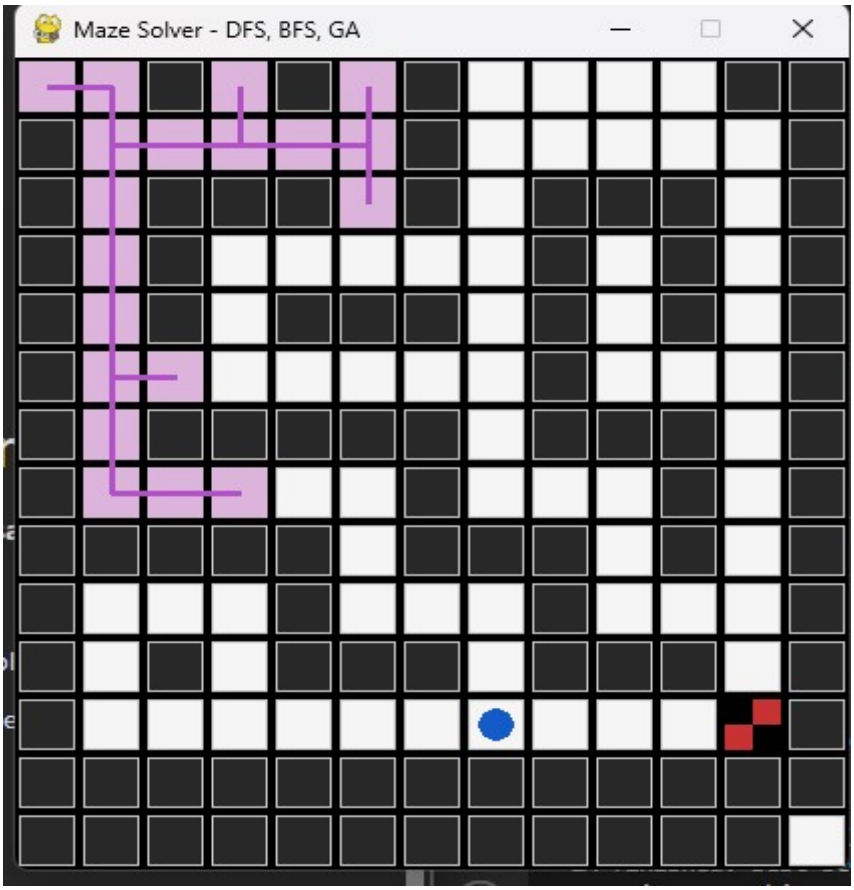
2. BFS search progress



3. DFS search progress



4. GA



Clear Comparison and Discussion of All Algorithms Used

Criterion	BFS	DFS	GA
Guarantees shortest path?	Yes	No	No (stochastic)
Deterministic?	Yes	Yes	No
Memory usage	High (queue)	Low (stack)	Moderate (population)
Time to solution (typical)	Moderate	Fast/variable	Varies (depends on gens)
Strengths	Finds minimal paths; predictable	Low memory; finds a path quickly	Flexible; can optimize noisy objectives; adaptable
Weaknesses	Higher memory, slower in large graphs	May explore long dead-ends; not shortest	Needs parameter tuning; runtime can be high; no optimality guarantee
Best use-case	Shortest-path in small/medium grids	Quick path discovery where path quality less important	Complex or partially observable domains; when heuristics are hard or when multiple objectives exist

Discussion

- BFS consistently finds the shortest path but uses more memory.
- DFS can get stuck in deep branches, producing longer paths.
- GA balances exploration and optimization, useful for more dynamic or large mazes.
- When applied to variant mazes, BFS remains reliable but slower, DFS changes path unpredictably, while GA adapts and converges to reasonable solutions over time.

Conclusion and Reflections

The project successfully integrates search algorithms within a visual maze environment.

It demonstrates practical differences between uninformed search (DFS/BFS) and evolutionary computation (GA).

Key lessons learned include:

- Importance of state representation and heuristic evaluation.
- Impact of population size and mutation rate on GA performance.
- Visualization as a useful tool for understanding algorithm behavior.