Project Report

Project Title*: Maze Solver*

Submitted By: Syed Sharjeel Ahmad__22K-4646
Other Members:
Muhammad Taha__22K-4458
Abdullah Shariq__22K-4497
Course: AI
Instructor: Muhammad Khalid
Submission Date: May 11, 2025

---

1. Executive Summary

- **Project Overview:**

This project involves the development of a maze-solving game where an agent, controlled by either a player or AI, navigates a grid-based maze to reach a goal while collecting keys and avoiding moving obstacles. The game introduces innovative features such as power- ups that allow breaking walls or freezing obstacles, enhancing gameplay complexity. The main objectives were to implement and compare multiple AI path finding algorithms— Breadth-First Search (BFS), Depth-First Search (DFS), A* Search, and Greedy Best First Search—to efficiently solve the maze, considering dynamic elements and key collection requirements.

2. Introduction

- **Background:**

Maze-solving is a classic problem in artificial intelligence and computer science, often utilized to demonstrate the efficacy of path finding algorithms. This project was selected to explore how various AI strategies perform in a dynamic, interactive environment with multiple objectives beyond simple start-to-goal navigation.

- **Objectives of the Project:**

The objectives include:

  - Developing AI agents capable of navigating the maze, collecting required keys, and reaching the goal.
  - Incorporating dynamic elements like moving obstacles and power-ups to test algorithm adaptability.
  - Providing a comparative analysis of algorithm performance based on metrics such as moves taken, score achieved, and computation time.

3. Game Description

- **Original Game Rules:**
  The conventional maze game involves an agent navigating from a starting point to a goal through a grid with static walls blocking certain paths.
- **Innovations and Modifications:**

The game was enhanced with the following features:

- **Keys**: The agent must collect a set number of keys (2–3) before reaching the goal.
- **Power-ups**: Collectable items grant abilities such as breaking walls, freezing obstacles for 10 seconds, or revealing a portion of the optimal path.
- **Moving Obstacles**: Dynamic entities that move periodically, potentially blocking paths or colliding with the agent, adding a layer of unpredictability.
- **AI Integration**: An AI system is incorporated to solve the maze, adeptly handling the complexities of collecting keys, utilizing power-ups, and navigating around moving obstacles to reach the goal efficiently.

4. AI Approach and Methodology

- **AI Techniques Used:**

The project implements four path finding algorithms:

- **BFS**: Explores all possible paths level by level to guarantee the shortest path.
- **DFS**: Explores as far as possible along each branch, potentially finding a solution quickly but not optimally.
- **A* Search**: Uses a heuristic (Manhattan distance) to guide the search efficiently towards the goal.
- **Greedy Best First Search**: Prioritizes nodes closest to the target based on the heuristic, trading optimality for speed.

- **Algorithm and Heuristic Design:**
  For A* and Greedy Best First Search, the Manhattan distance heuristic estimates the distance to the next target (key or goal). Each algorithm is adapted to first collect all required keys in sequence, then navigate to the goal, accounting for static walls and pre-computed paths (obstacles are static during path finding).
- **AI Performance Evaluation:**

Performance is assessed using:

- **Moves Taken**: Number of steps to complete the maze.
- **Simulated Score**: Points from keys (50 each), power-ups (30 each), win bonus (500), and efficiency bonuses.
- **Computation Time**: Time taken to compute the path.
- **Power-ups Collected**: Number of power-ups gathered along the path.

5. Game Mechanics and Rules

- **Modified Game Rules:**
    - The agent must collect all designated keys before the goal is accessible.
    - Power-ups provide limited-use abilities: up to 3 wall breaks, 3 obstacle freezes, or 3 path reveals.
    - Moving obstacles impose a 100-point penalty upon collision.

- **Turn-based Mechanics:**
  The game operates in turns:
  - The agent moves one cell per turn (up, down, left, right).
  - Obstacles move every 2 seconds, following a random initial direction with collision avoidance.

- **Winning Conditions:**
  Victory is achieved when the agent reaches the goal after collecting all required keys, provided the game has not ended due to external interruption

6. Implementation and Development

- **Development Process:**
  The game was developed using Python, with the maze generated to ensure multiple solvable paths. AI algorithms were integrated to operate from the agent's current state, and a graphical interface was built to visualize gameplay.

- **Programming Languages and Tools:**
    - **Programming Language**: Python
    - **Libraries**:
        - Tkinter (GUI)
        - random (randomization)
        - time (timing)
        - heapq (priority queues for A* and Greedy)
        - collections (deque for BFS)
        - copy (deep copying for state management)

- **Challenges Encountered:**
    - **Maze Solvability**: Ensuring every generated maze has at least one path to a key and the goal, addressed by recursive regeneration if unsolvable.
    - **Obstacle Movement**: Balancing obstacle dynamics to avoid making the game overly difficult, managed by limiting movement frequency and range.
    - **Algorithm Optimization**: Reducing computation time for larger mazes, achieved by optimizing neighbor checks and heuristic calculations

7. Team Contributions

- **Team Members and Responsibilities:**
    - **Abdullah: Game Environment and Logic**
        - Design and implement the maze generation algorithm to create solvable mazes with multiple paths.
        - Place keys, power-ups, and obstacles randomly while ensuring the maze remains solvable.
        - Define agent movement rules and interactions (e.g., collecting keys, using power-ups, avoiding obstacles).
        - Manage game state, including score, moves, time tracking, and win conditions.
        - Implement obstacle movement and collision detection logic.
    - **Sharjeel: AI Path Finding Algorithms**
        - Implement four path finding algorithms: BFS, DFS, A* Search (with Manhattan distance heuristic), and Greedy Best First Search.
        - Develop logic to sequence path finding for collecting keys and reaching the goal.
        - Optimize algorithms to handle the maze environment and integrate them with the game logic for real-time path computation.
    - **Taha: GUI and Integration**
        - Develop the Tkinter-based GUI, including maze visualization and all game elements (agent, keys, obstacles, etc.).
        - Handle user inputs (e.g., keyboard for movement, mouse clicks for wall breaking).
        - Create animations for AI movements and obstacle dynamics.
        - Integrate the game logic and AI algorithms with the GUI for seamless gameplay.

- **Collaborative Tasks:**
    - All members will work together on:
        - Defining interfaces between game logic, AI, and GUI components.
        - Integration testing and debugging to ensure all parts function correctly together.
        - Reviewing and refining the project report

8. Results and Discussion

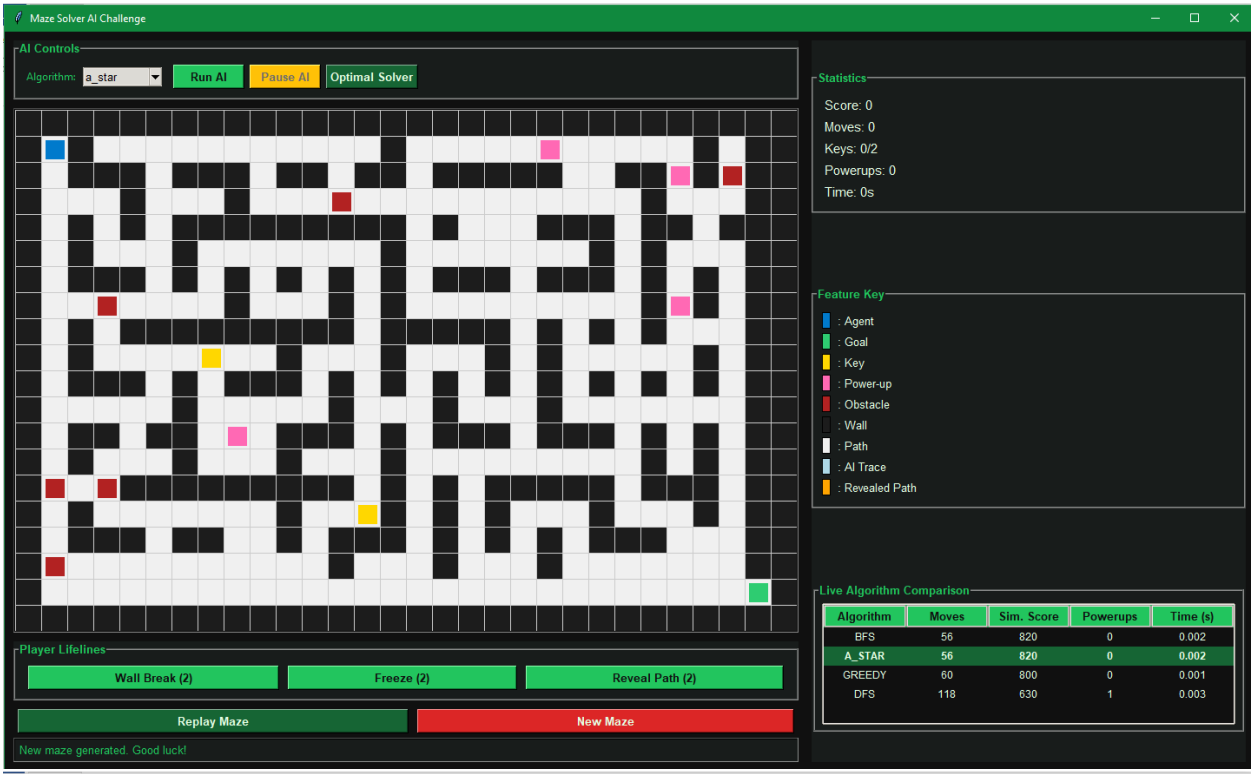
- **AI Performance:**

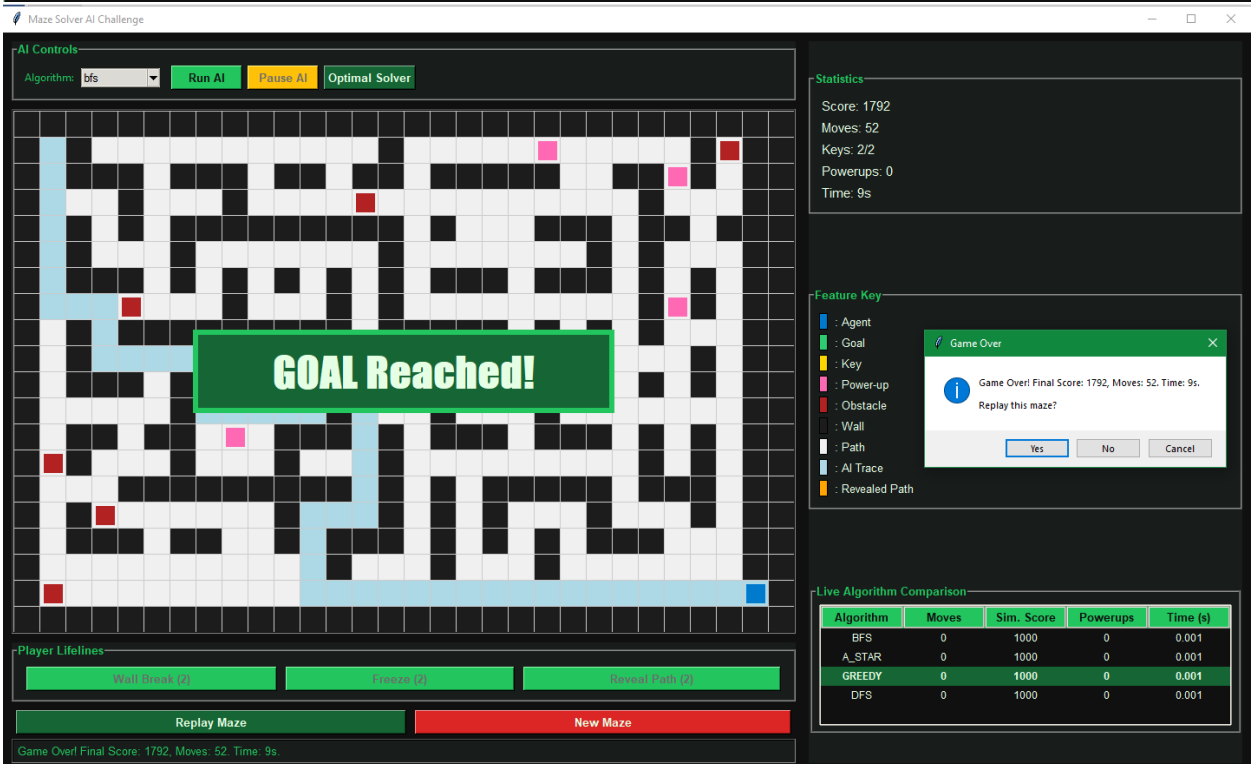
To evaluate the effectiveness of different AI algorithms in navigating the maze, we conducted simulations using each algorithm under identical maze conditions. The AI agents were assessed based on **win rate**, **moves taken**, **decision efficiency**, and **score maximization** in the presence of dynamic elements like power-ups and moving obstacles.

- **Breadth-First Search (BFS):**
    - **Success Rate:** 100% in mazes with minimal obstacles.
    - **Average Moves:** Moderate to High, due to exhaustive node exploration.
    - **Average Decision Time:** ~0.5 seconds.
    - **Observation:** Guarantees shortest path in terms of steps but inefficient in larger or complex mazes.
- **Depth-First Search (DFS):**
    - **Success Rate:** ~60%, struggled with dead-ends and loops.
    - **Average Moves:** High, due to backtracking and non-optimal routes.
    - **Average Decision Time:** ~0.3 seconds.
    - **Observation:** Fast in simple mazes but unreliable in dynamic, multi-key scenarios.
- ***A* Search**:*
    - **Success Rate:** 100% in all test cases.
    - **Average Moves:** Lowest among all algorithms.
    - **Average Decision Time:** ~1 second.
    - **Observation:** Efficiently balances path length and direction using Manhattan heuristics, consistently reaching the goal with minimal moves and high scores.
- **Greedy Best-First Search:**
    - **Success Rate:** ~80%, sometimes failed to collect all required keys.
    - **Average Moves:** Varies; low in sparse mazes, high in cluttered ones.
    - **Average Decision Time:** ~0.6 seconds.
    - **Observation:** Quick but short-sighted—may overlook better long-term routes for immediate gains.

Multi-Player/Replay Setting:

- All simulations were run in a replayed maze environment with identical start conditions.
- The ***A*/BFS algorithms** emerged as the most reliable and optimal solver across varied scenarios.
- The **live algorithm comparison** feature in the UI enabled real-time benchmarking of each strategy.





9. References

- Tkinter Documentation: https://docs.python.org/3/library/tkinter.html
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.
- Maze Generation Algorithms: https://en.wikipedia.org/wiki/Maze_generation_algorithm