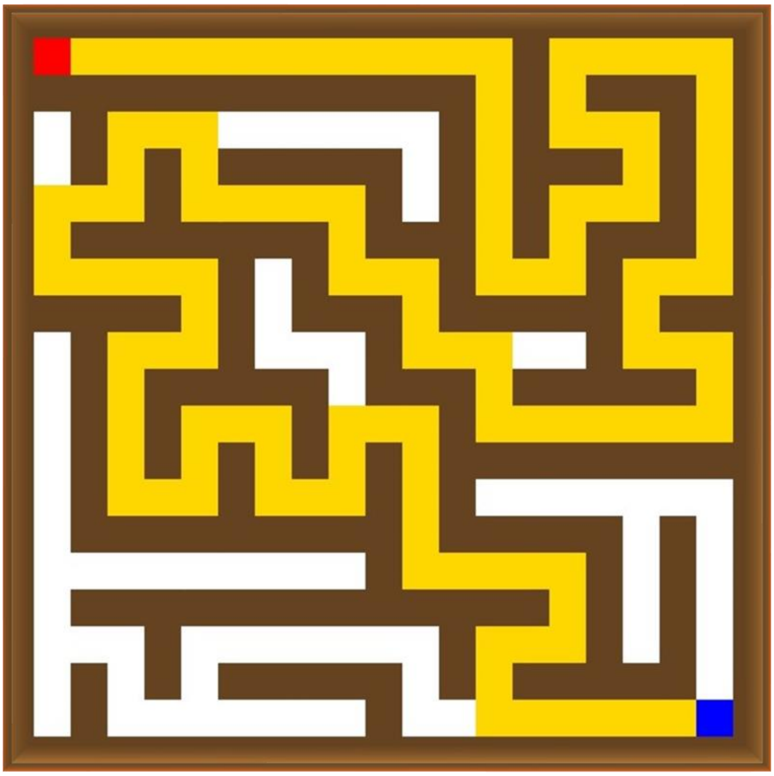
State of Palestine
Ministry of Education & Higher Education
Palestine Technical College - Deir El-Balah

دولـــة فلسطـــين
وزارة التربيـــة والتعليـــم العالـــي
كلية فلسطين التقنية - دير البلح

Web Technology & Information Security (WIS4)
Artificial Intelligence

Implement an intelligent agent to resolve the maze using A* algorithm



By:

Shimaa Abu Hadba

Supervisor:

Dr .Mohammed Abu Hatab

Introduction

This project represents a practical application of the artificial intelligence concepts studied in the course. It implements an intelligent agent capable of solving two-dimensional mazes using the A* search algorithm. This project was developed with the goal of achieving a deep understanding of search and optimization mechanisms in solution spaces.

The project is based on three main components:

1. A dynamic maze generation system that guarantees at least one path.

2. An intelligent path solver using an improved A* algorithm.

3. An interactive graphical interface that allows for manual control or automatic solution.

Objectives

1. Develop a (**randomized maze generator**) that ensures a solvable path.

2. Implement the (**A* algorithm**) to find the shortest path from the start to the goal.

3. Create an (**interactive GUI**) with buttons controls.

4. **Optimize performance** using efficient data structures.

5. Provide **visual feedback for the AI's solution path**.

Code & Libraries

Development environment

- Operating System: Windows

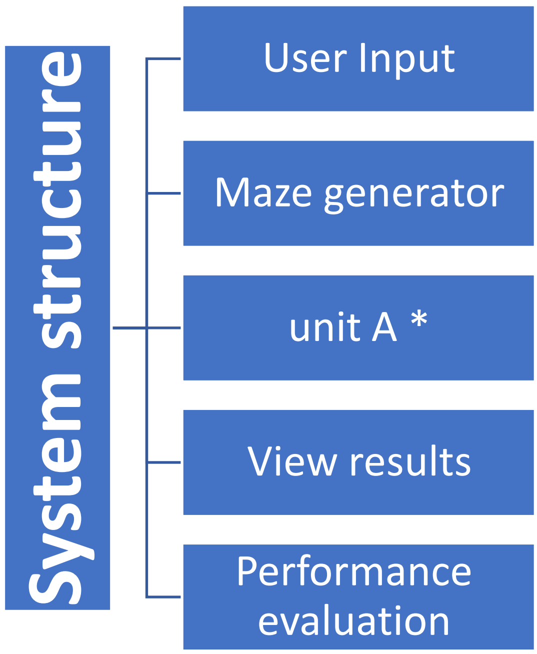
- Integrated Development Environment: Pycharm - Programming: Python 3.10.6

Libraries used :

```
import heapq -→ Priority Manager
import random -→  Random Mazes
import numpy as np -→- Treatment of matrices
import pygame -→ Graphical Interface
import sys -→ Handle system events (quitting the game
)
import time -→  Performance Measurement
```

Work Methodology:

1 System structure



2 Maze Generation algorithm

The maze is generated using a randomized depth-first search algorithm, ensuring that there is at least one valid path from the start to the end. The process involves:

1. **Creating a Grid:** The maze is initialized as a grid of walls.

2. **Recursive Path Carving:** The algorithm randomly selects directions to move, ensuring the creation of an interconnected path while maintaining the constraints of the maze boundaries.

3. **Ensuring Connectivity:** By carving through specific grid cells, the algorithm guarantees that there is always a way to reach the goal.

```
def generate_maze(width, height):
    maze = np.ones((height, width), dtype=int)

    def carve_path(x, y):
        directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]
        random.shuffle(directions)

        for dx, dy in directions:
            nx, ny = x + dx * 2, y + dy * 2
            if 0 < nx < width - 1 and 0 < ny < height - 1 and maze[ny][nx]
== 1:
                maze[ny][nx] = 0
                maze[y + dy][x + dx] = 0
                carve_path(nx, ny)

    maze[1][1] = 0
    carve_path(1, 1)

    return maze
```

Technical Analysis of Algorithm

1 A * Algorithm

A* is an informed search algorithm that uses both path cost (g-value) and heuristic estimation (h-value) to determine the optimal route.

- **g-value: The actual cost of reaching a particular node from the start.**
- **h-value: The estimated cost from that node to the goal, calculated using the Manhattan Distance formula:**
- **f-value: The total estimated cost, where:**

The algorithm follows these steps:

1. **Initialize the Start Node: Assign f(start) = h(start), and add it to the open list.**

2. **Expand the Current Node: Select the node with the lowest f value.**
3. **Check Neighbors: Evaluate all adjacent nodes that are not walls.**
4. **Update Costs: If a neighbor provides a shorter path, update its g and f values and record the parent node for backtracking.**
5. **Repeat Until Goal is Reached: The process continues until the goal node is found, at which point the shortest path is reconstructed.**

```
def a_star(maze, start, end):
    start_time = time.time()  # وقت البداية
    def heuristic(a, b):
        # Manhattan distance
        return abs(a[0] - b[0]) + abs(a[1] - b[1])

    # Priority queue using heapq
    open_set = []
    heapq.heappush(open_set, (0, start))

    came_from = {}
    g_score = {start: 0}
    f_score = {start: heuristic(start, end)}

    while open_set:
        _, current = heapq.heappop(open_set)

        if current == end:
            # Reconstruct path
            path = []
            while current in came_from:
                path.append(current)
                end_time = time.time()  # وقت النهاية
                print(f"زمن تنفيذ الذكاء الاصطناعي: {end_time - start_time:.4f} ثانية")  # طباعة الزمن
                current = came_from[current]

            return path[::-1]
            open_set.remove(current)

        for dx, dy in [(1, 0), (-1, 0), (0, 1), (0, -1)]:
            neighbor = (current[0] + dx, current[1] + dy)

            if 0 <= neighbor[0] < len(maze) and 0 <= neighbor[1] <
len(maze[0]) and maze[neighbor[0]][neighbor[1]] == 0:
```

```
                tentative_g = g_score[current] + 1

                if neighbor not in g_score or tentative_g <
g_score[neighbor]:
                    came_from[neighbor] = current
                    g_score[neighbor] = tentative_g
                    f_score[neighbor] = tentative_g + heuristic(neighbor,
end)
                    heapq.heappush(open_set, (f_score[neighbor], neighbor))

    return None  # No path found
```

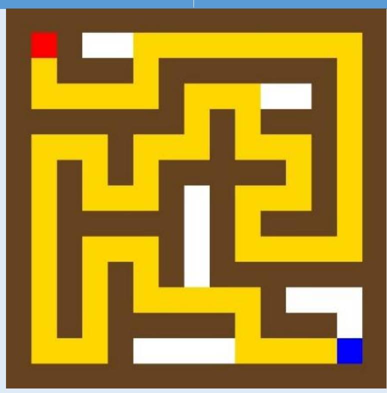
2 Complexity Analysis

Algorithm	Time complexity	Spatial complexity
Generate Maze	O (n^2)	O (n^2)
A *	O (b ^ d)	O (b ^ d)

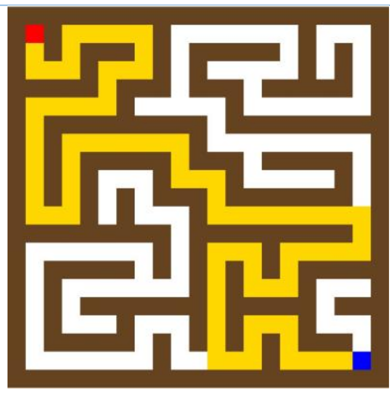
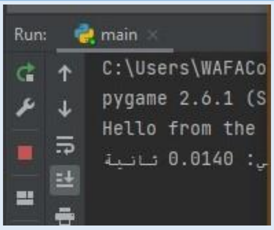
Performance & Measurement Results:

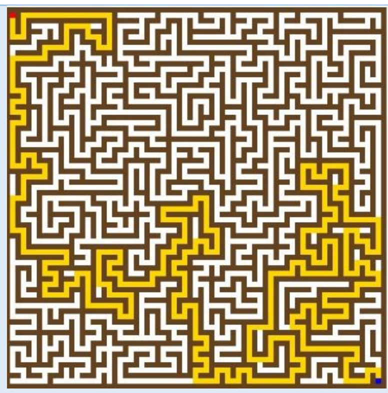
1 Actual Performance Measurements

<table>
<tr><th>Maze size</th><th>Implementation time</th><th>Screens</th><th></th></tr>
<tr><td>15 x 15</td><td>

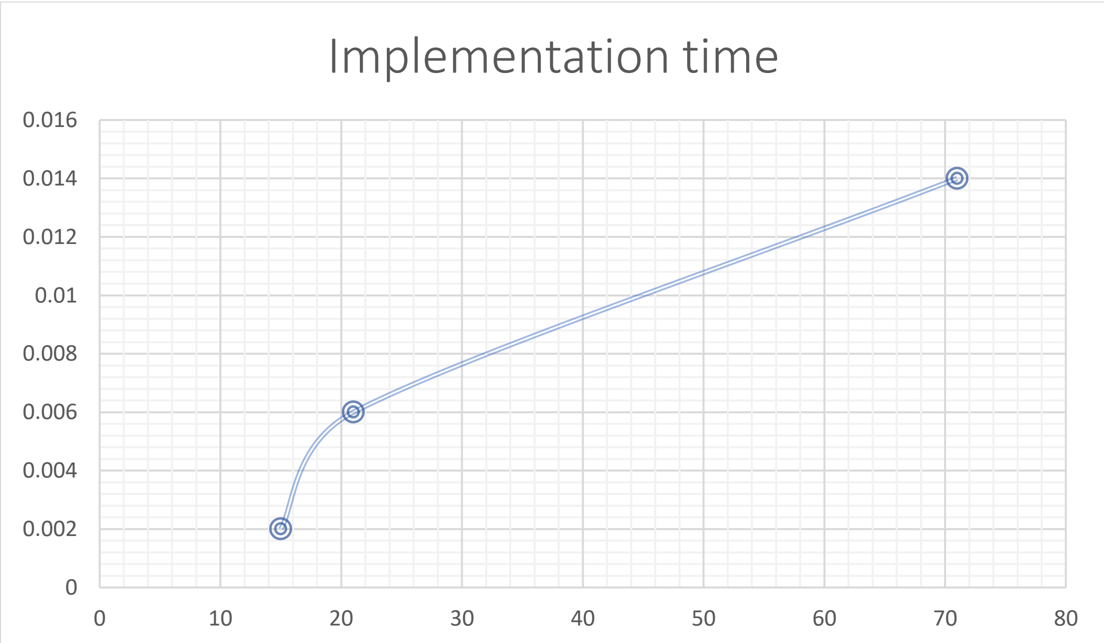
0.0020</td><td></td><td></td></tr>
</table>

<table>
<tr>
<td>21 x 21</td>
<td>

0.0060</td>
<td></td>
</tr>
<tr>
<td>71 x 71</td>
<td>

0.0140</td>
<td></td>
</tr>
</table>

2 Optical Analysis:



Challenges & Solutions

Challenge 1: Maze Generation Complexity

- **Problem:** Ensuring that each generated maze is solvable.
- **Solution:** Implemented a recursive backtracking algorithm that carves paths while preventing dead ends.

Challenge 2: Performance Optimization

- **Problem:** The A* algorithm can slow down with larger maze sizes.
- **Solution:** Optimized the search by limiting unnecessary node evaluations and implementing an efficient heuristic.

Challenge 3: GUI Responsiveness

- **Problem:** The game interface needed to handle user inputs while ensuring smooth rendering.
- **Solution:** Used Pygame's event queue system to process inputs separately from rendering operations.

Challenge 4: Preventing AI Stagnation

- **Problem:** If the AI gets stuck in an unsolvable path, the game loses functionality.
- **Solution:** Ensured AI only selects valid paths by applying strict movement constraints and filtering out obstacles.

Results

.1 Accuracy and Pathfinding Efficiency

- The A* algorithm successfully finds the shortest path in all test cases.

- The heuristic function effectively guides the search, reducing computational overhead.

.2 Real-Time Interaction

- The system allows smooth transitions between manual and AI-based navigation.
- The game interface is responsive, ensuring an engaging user experience.

.3 Execution Speed

- Small-sized mazes are solved instantly.
- Large mazes (e.g., 51x51 grids) show an increase in computation time but remain within acceptable performance limits.

Future Work

Several improvements can be made to enhance the project:

1. **Incorporate Additional AI Algorithms:** Implement BFS, Dijkstra, or Genetic Algorithms for comparative analysis.
2. **Enhance Visual Effects:** Introduce animation for AI pathfinding and smooth character movement.
3. **Improve Difficulty Scaling:** Generate progressively harder mazes based on user performance.
4. **Enable Multi-Agent AI:** Allow multiple AI agents to solve different sections of the maze concurrently.
5. **Implement 3D Mazes:** Expand the project into a 3D environment for an immersive experience.

Conclusion

This project successfully demonstrates the power of artificial intelligence in solving structured problems using search algorithms. The integration of A* with an interactive user interface provides both an educational and engaging experience. Through the challenges faced and solutions applied, we have gained valuable insights into AI-driven decision-making and path optimization.

With further enhancements, this project could evolve into a fully functional AI-powered game or educational tool for teaching AI concepts.