UCS2504: FOUNDATIONS OF ARTIFICIAL INTELLIGENCE

Project Report

Application using AI algorithm

Maze Runner: A * Pursuit

Team members:

Priyaadharshini R (CSE-B) - 3122215001072

Pranaya P (CSE-B) - 3122215001064

Prahalad R (CSE-B) - 3122215001067

Problem Statement:

Objective:

Develop a grid-based game that demonstrates the A* pathfinding algorithm in a Python environment using Pygame. The game features an agent (player-controlled character), an enemy (AI-controlled character using A* for movement), and a goal position. The agent's objective is to reach the goal without being caught by the enemy.

Solution:

Implement a turn-based game where the player moves the agent each turn, and the enemy moves toward the agent using A* pathfinding. The game ends when the agent reaches the goal or is captured by the enemy.

Software Requirements Specification:

Functional Requirements:

- *User Interaction:* The player should be able to control the agent using keyboard inputs.
- *AI Movement:* The enemy should autonomously move towards the agent using the A* algorithm.
- *Win and Lose Conditions:* The game should detect and respond to win (reaching the goal) and lose (enemy captures the agent) conditions.
- *Graphical Representation:* Visual representation of the game grid, agent, enemy, and goal.
- *End-Game Effects:* Display messages and effects (like confetti) at the end of the game.

Non-Functional Requirements:

- *Performance:* The game should run smoothly with minimal latency in user input and AI response.
- *Reliability:* The A* algorithm should consistently find the most efficient path to the agent.
- *Usability:* The game should be easy to understand and play, with intuitive controls and clear visual elements.
- *Maintainability:* The code should be well-organized and documented for easy updates and maintenance.

Design Alternatives:

- Pathfinding Algorithm:
    - Primary: A* Algorithm for its efficiency and accuracy in grid-based pathfinding.
    - Alternative: Dijkstra's Algorithm for simpler but potentially less efficient pathfinding.
- Game Interface:
    - Primary: Pygame for an interactive and graphical interface.
    - Alternative: Command-line interface for a simpler implementation.
- Agent and Enemy Mechanics:
    - Primary: Turn-based movement.
    - Alternative: Real-time movement for a more dynamic gameplay experience.

Algorithm: *Maze Runner: A * Pursuit*

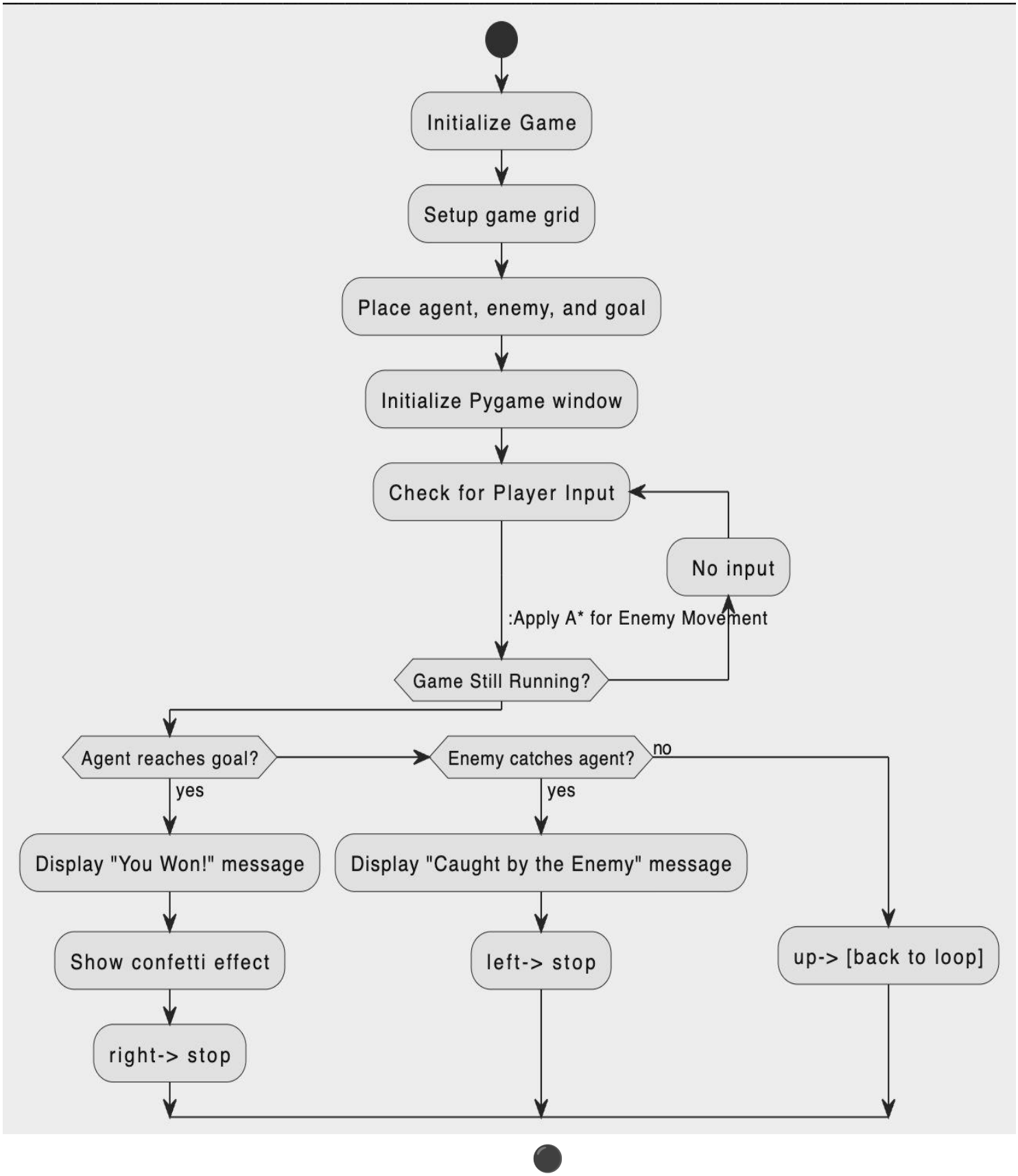
1. Initialize Game
    1.1. Create a grid-based map
    1.2. Randomly place the agent, enemy, and goal on the map
    1.3. Ensure the initial positions are valid and meet the game's conditions
    1.4. Load necessary resources (images, sounds, etc.)

2. While the game is running
    2.1. If player input is detected
        2.1.1. Move the agent based on the input
        2.1.2. If the agent's new position is the goal
            2.1.2.1. Display "You Won!" message
            2.1.2.2. Trigger confetti effect
            2.1.2.3. Exit the game loop

    2.2. Apply the A* pathfinding algorithm for the enemy
        2.2.1. Calculate the path from the enemy to the agent
        2.2.2. Move the enemy along the calculated path
        2.2.3. If the enemy reaches the agent's position
            2.2.3.1. Display "Caught by the Enemy" message
            2.2.3.2. Exit the game loop

    2.3. Render the updated positions of the agent, enemy, and goal on the grid
3. End While
4. Close the game and release resources
5. End Algorithm.

Design:


Initialize Game
Setup game grid
Place agent, enemy, and goal
Initialize Pygame window
Check for Player Input
No input
:Apply A* for Enemy Movement
Game Still Running?
Agent reaches goal?
Enemy catches agent?
no
yes
yes
Display "You Won!" message
Display "Caught by the Enemy" message
Show confetti effect
left-> stop
up-> [back to loop]
right-> stop

System Architecture:

1. **Overall Structure:**
    - The game is structured around a modular and layered architecture, ensuring separation of concerns and ease of maintenance.

2. **Core Modules:**
    - *Game Logic Module:*
      Manages the rules of the game, including the initialization of the game state, updating the game based on player actions, and determining win/lose conditions.
    - *Pathfinding Module:*
      Dedicated to implementing the A* algorithm. This module takes the current positions of the enemy and the agent and calculates the optimal path for the enemy.
    - *Input Handling Module:*
      Interprets and processes player inputs, translating them into movements or actions in the game.

3. **Data Management:**
    - *State Management:*
      Maintains the state of the game, including the positions of the agent, enemy, and goal, as well as the state of the game grid.
    - *Resource Management:*
      Manages the loading and unloading of resources such as images, fonts, and sound effects.

4. **Interface and Communication:**
    - Each module communicates with others through well-defined interfaces. For instance, the game logic module sends commands to the rendering module about what to draw based on the game's current state.

5. **User Interface Layer:**
    - Responsible for all elements of the game's user interface, including displaying the game grid, menus, and end-game screens.
    - Ensures responsiveness and usability, providing immediate visual feedback to player actions.

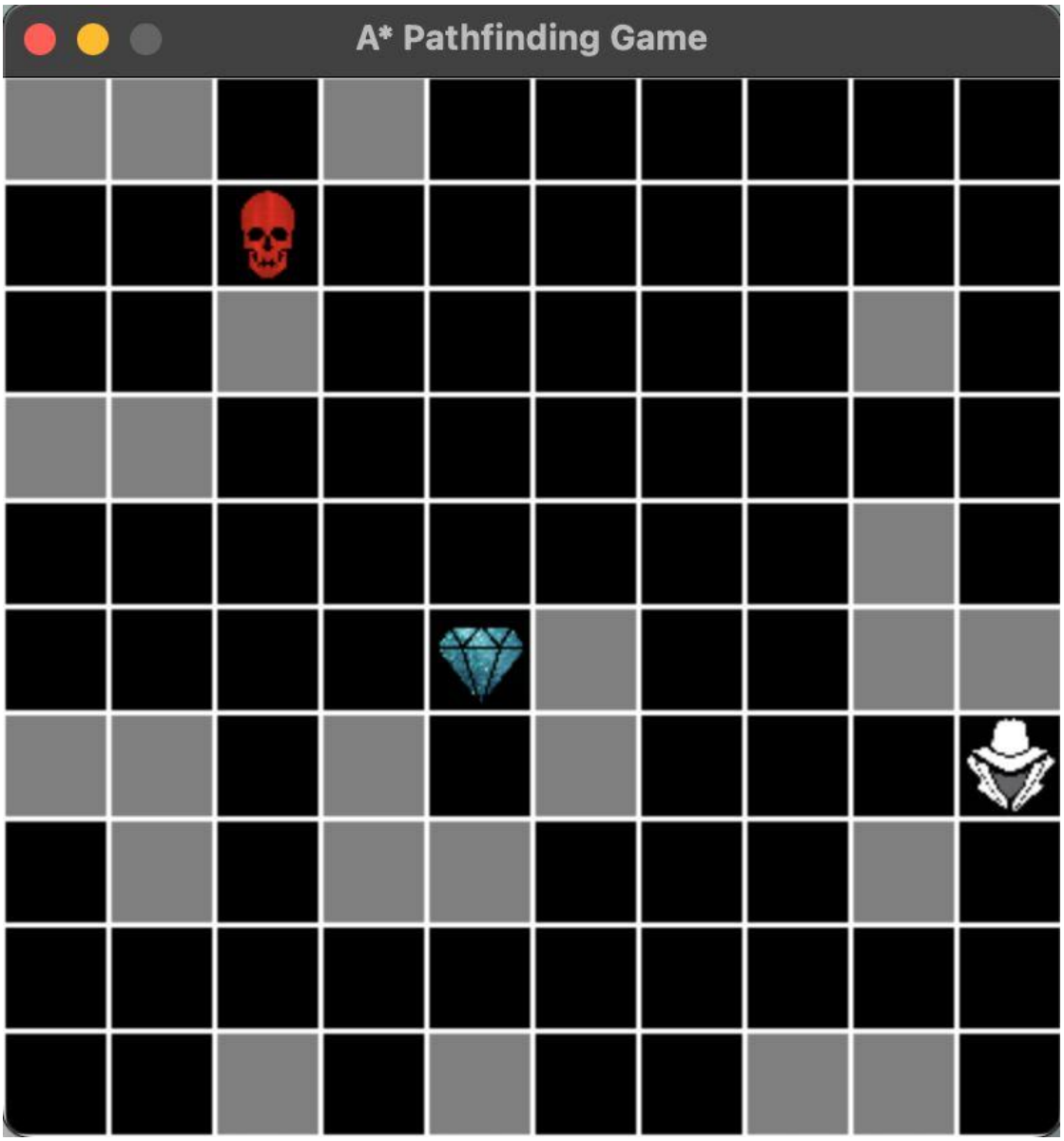
6. **AI and Pathfinding Layer:**
    - Implements the logic for the enemy's AI using the A* pathfinding algorithm.
    - Makes decisions based on the game's current state and provides instructions for enemy movement.

7. **Game Control Layer:**
    - Acts as the central coordinator for the game.
    - Manages the game loop, orchestrating the flow of the game, and coordinates between user inputs, game logic, AI decisions, and rendering.

8. **External Libraries and Frameworks:**
    - Utilises Pygame for rendering and handling user input.
    - May integrate other libraries for additional functionalities, like sound handling.

State Space:

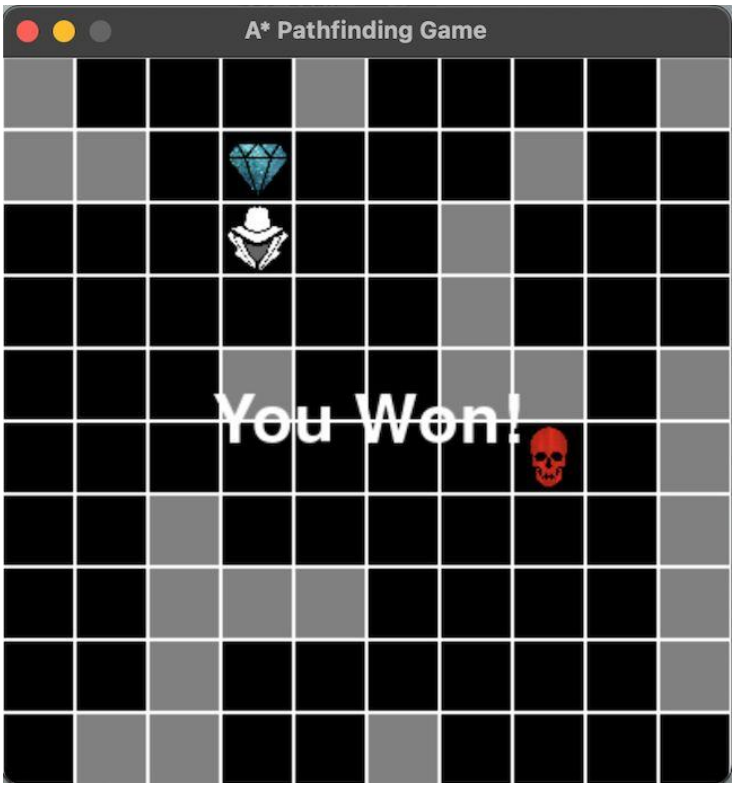
The grid map is represented as a 2D array, with different elements (agent, enemy, walls, open spaces, goal).



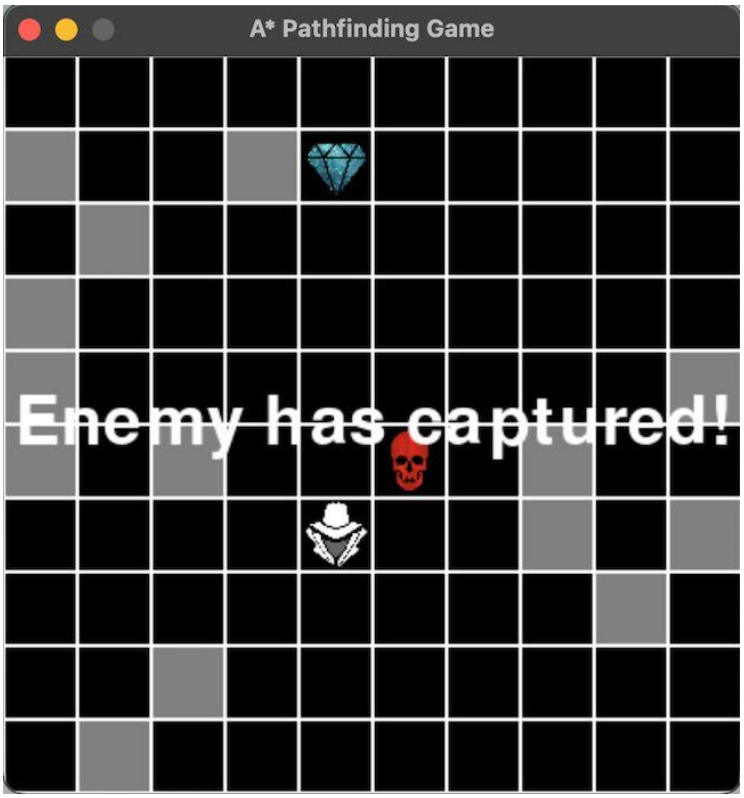
Test Cases:

Various grid configurations to test the effectiveness and reliability of the A* algorithm.

Agent winning the game:



Enemy capturing the agent:



Methodology:

- Initialize Game Environment: Create a grid map and randomly place the agent, enemy, and goal.
- User Input Handling: Allow the player to move the agent each turn.
- A Pathfinding*: Implement the A* algorithm for the enemy's movement towards the agent.
- Detect Game End Conditions: Check for win/lose states at the end of each turn.
- Graphical Rendering: Use Pygame to render the game state visually.

Validation:

Validation involves verifying that the game meets its intended requirements and specifications. It includes testing the game's functionality, performance, usability, and overall player experience.

Functional Testing:

- *Pathfinding Accuracy:*
  Validate that the A* algorithm correctly navigates the enemy towards the agent.
- *Player Input Response:*
  Ensure that the agent moves as expected according to player inputs.
- *Win/Lose Conditions:*
  Test for correct detection of win (reaching the goal) and lose (enemy catching the agent) conditions.

Usability Testing:

- *User Interface:*
  Assess the intuitiveness and ease of use of the game interface.
- *Player Feedback:*
  Collect feedback from test players regarding the game's playability and enjoyment.

Performance Testing:

- *Game Speed and Responsiveness:*
  Check the game runs smoothly without lags, especially during pathfinding calculations and player movements.
- *Resource Usage:* Monitor the game's consumption of system resources to ensure it doesn't overuse memory or processing power.

Compatibility Testing:

- *Platform Testing*:
  If applicable, test the game on different platforms or Python versions to ensure compatibility.

Best Practices:

1. **Modular Programming**
   Purpose: Break down the game's code into separate, independent modules (like player movement, enemy AI, and game rendering).
   Benefit: Enhances readability, maintainability, and scalability of the code.

2. **Test-Driven Development (TDD)**
   Purpose: Write tests for each function or module before implementing them.
   Benefit: Ensures reliability and reduces the likelihood of bugs in the code.

3. **User-Centric Design**
   Purpose: Focus on user experience in the game design, considering the user interface, graphics, and interaction.
   Benefit: Increases user engagement and satisfaction.

4. **Iterative Development and Feedback Integration**
   Purpose: Develop the game in iterations, integrating user feedback at each stage.
   Benefit: Ensures the game meets user expectations and can adapt to user needs.

5. **Error Handling**
   Purpose: Implement robust error handling and maintain logs for unexpected behaviours or crashes.
   Benefit: Helps in quick debugging and ensures the game's stability.

Technological Improvement:

This project utilises Weak AI:
"Maze Runner: A* Pursuit" utilises Weak AI due to its focus on specific tasks within a defined environment and the absence of generalised intelligence or learning capabilities. The AI, particularly the enemy character using the A* pathfinding algorithm, is programmed for the sole purpose of navigating a maze to catch the player. It lacks consciousness, self-awareness, and the ability to adapt or learn from experience. These attributes align with the characteristics of Weak AI, which is designed for narrow, rule-based tasks without exhibiting broader, human-like cognitive abilities.

Scalability and Future Enhancements:

- The design should allow for easy addition of new levels, challenges, or features like power-ups or additional enemies.
- Consideration for potential multiplayer features in the future

Inferences:

Agent (X, Y)	Diamond (X, Y)	Enemy (X, Y)	Outcome
(3, 7)	(0, 1)	(7, 0)	Win
(5, 1)	(2, 6)	(7, 4)	Win
(4, 6)	(9, 9)	(7, 7)	Win
(4, 8)	(2, 5)	(0, 7)	Loss
(9, 7)	(0, 5)	(2, 6)	Win
(8, 8)	(0, 6)	(1, 8)	Win
(8, 6)	(9, 5)	(2, 9)	Loss
(8, 8)	(7, 1)	(3, 8)	Win
(9, 7)	(6, 7)	(3, 6)	Loss
(6, 7)	(2, 0)	(4, 3)	Win
(7, 8)	(3, 3)	(9, 3)	Win
(2, 2)	(6, 9)	(4, 7)	Loss
(9, 7)	(1, 6)	(2, 9)	Win
(6, 8)	(5, 0)	(5, 3)	Loss
(3, 3)	(5, 7)	(6, 2)	Win
(3, 4)	(2, 7)	(3, 7)	Loss

Based on the table provided with both "Win" and "Loss" outcomes from multiple runs of the game, we can derive the following inference:

- **Game Outcomes Vary:**
  The game outcomes are not always the same. Some runs result in a "Win" for the player, while others result in a "Loss" where the enemy captures the player.

- **Randomness Plays a Role:**
  The game involves randomness in generating the initial positions of the agent, diamond, and enemy. This randomness can lead to different game scenarios.

- **Player's Strategy:**
  The player's movements, controlled by arrow keys, can impact the outcome. The player's strategy in avoiding the enemy and reaching the goal (diamond) appears to be a critical factor.

- **Difficulty Level:**
  The difficulty of the game can vary depending on the initial positions. Runs with more open paths may result in wins, while runs with confined spaces or unfavourable initial positions may result in losses.

- **Balancing Act:**
  The game's challenge seems to be finding a balance between reaching the goal (diamond) while avoiding the enemy. Timing and positioning are crucial for success.

- **Random Elements:**
  The placement of walls, the locations of the agent, enemy, and diamond, and the movement of the enemy are all influenced by random factors, adding an element of unpredictability to each run.

- **Player's Skill:**
  The player's ability to navigate the grid and make strategic moves significantly impacts the outcome. With practice and experience, the player may improve their chances of winning

Learning Outcomes:

- Understood and implemented the A* pathfinding algorithm.
- Development of a basic game using Pygame in Python.
- Insights into AI applications in game development.
- Experience in designing and testing a software application.