ESCAPE AI HUNTERS

MINI PROJECT REPORT

Submitted by

PRABAKARAN K (9517202309085)

SANJAY K N (9517202309101)

JAYAVELAN S (9517202309041)

NANDHAGOPAL V (9517202309074)

in

23AD402 – ARTIFICIAL INTELLIGENCE

DURING

IV SEMESTER : 2024 - 2025

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

MEPCO SCHLENK ENGINEERING COLLEGE

SIVAKASI

MAY 2025

MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

(An Autonomous Instituion Affliated to Anna University)

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

BONAFIDE CERTIFICATE

Certified that this project report titled **ESCAPE AI HUNTERS** is the bonafide work of **Jayavelan S (9517202309041), Nandhagopal V (9517202309074), Sanjay K N (9517202309101), Prabakaran K (95172020309085)** who carried out this work under my guidance for the course **"23AD402: Artificial Intelligence"** during the Fourth semester.

SIGNATURE

Dr. Shenbagarajan Anantharajan,
Associate professor,
Department of Artificial Intelligence and Data Science.
Mepco Schlenk Engineering College (Autonomous)
Sivakasi.

SIGNATURE

Dr. J. Angela Jennifa Sujana,
Professor & Head of department,
Department of Artificial Intelligence and Data Science.
Mepco Schlenk Engineering College (Autonomous)
Sivakasi.

Submitted for viva-Voice Examination held at **MEPCO SCHLENK ENGINEERING COLLEGE (Autonomous), SIVAKASI** on **_____.**

ACKNOWLEDGEMENT

First and foremost, we express our deepest gratitude to the LORD ALMIGHTY for His abundant blessings, which have guided us through our past, present, and future endeavors, making this project a successful reality.

We extend our sincere thanks to our management, esteemed Principal, Dr.S.Arivazhagan, M.E., Ph.D., for providing us with a conducive environment, including advanced systems and well-equipped library facilities, which have been instrumental in the completion of this project.

Our heartfelt appreciation goes to our project guides Dr. Shenbagarajan Anantharajan, Associate Professor, Mrs. A. Antony Shyamalin Assistant Professor (Sl.Grade), Department of Artificial Intelligence and Data Science for their unwavering support, insightful suggestions, and expert guidance. Their experience and encouragement have been vital in shaping our project and helping us bring forth our best.

We are profoundly grateful to our Head of the Artificial Intelligence and Data Science Department, Dr. J. Angela Jennifa Sujana, M.Tech., Ph.D., for granting us this golden opportunity to work on a project of this significance and for her invaluable guidance and encouragement throughout.

We also extend our sincere gratitude to all our faculty members, lab technicians, and our beloved family and friends, whose timely assistance and moral support played a pivotal role in making this mini-project a success

ABSTRACT

Escape AI Hunters is a dynamic and interactive mini-project developed using Python and Pygame that merges strategic planning, real-time action, and intelligent pathfinding. Set in a 50x50 grid maze, the game challenges players to reach a randomly chosen destination tile while being pursued by two AI-controlled bots. These bots utilize the A* search algorithm to dynamically calculate the shortest path to the player based on their real-time movements, ensuring unpredictable and engaging gameplay.

The maze is populated with static obstacles in the form of walls, requiring the player to navigate thoughtfully. The interface is visually intuitive, with color-coded representations for players (green), bots (red), walls (black), and gridlines, enabling players to clearly distinguish game elements. Players can move in both cardinal and diagonal directions using intuitive controls, enhancing movement flexibility and strategic options. Real-time feedback such as visual updates and animations keeps the experience immersive.

The replayability of *Escape AI Hunters* is high due to the randomized destination tile in each session and the adaptive behavior of the bots, which recalibrate paths with every player move. This combination of adaptability and randomness ensures no two games are alike, pushing players to continuously improve their tactics. The game is accessible for users of all ages, offering a simple control scheme alongside a rich layer of strategic depth.

Beyond entertainment, the project demonstrates practical implementation of AI pathfinding algorithms, making it a useful educational tool for understanding concepts like heuristic search and real-time decision-making. Overall, *Escape AI Hunters* provides a balanced blend of challenge and fun, making it suitable for both casual gameplay and algorithm enthusiasts.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 PROBLEM STATEMENT

The Escape AI Hunters Game addresses the problem of creating an engaging, algorithmdriven interactive experience where players must navigate a maze while being pursued by intelligent bots. The challenge lies in designing a system that dynamically adapts to the player's movements, ensuring an unpredictable and strategically challenging gameplay experience. Players must navigate a 50x50 grid maze filled with obstacles, aiming to reach a randomly assigned destination tile while avoiding capture. The bots, powered by the A* search algorithm, dynamically calculate optimal paths to chase the player, adapting in real-time as the player moves. This combination of real-time decisionmaking, adaptive pathfinding, and maze navigation presents a complex problem that requires careful integration of efficient algorithms and game mechanics.

Additionally, the problem involves ensuring a visually appealing interface and userfriendly controls to maintain immersion and engagement. The maze must be clearly represented, with distinct visual elements for walls, bots, players, and the destination tile. The game must provide smooth controls for the player and responsive visual feedback during gameplay. Balancing computational efficiency with an intuitive user experience is essential to delivering a rewarding game. The random generation of destination tiles and the bots' adaptive behavior ensure high replayability, making the game enjoyable for a wide range of users while showcasing the practical application of pathfinding algorithms in an interactive and entertaining context.

1.2 OBJECTIVES

• **Interactive Gameplay**: Develop a maze-based game where players navigate through a grid while being pursued by intelligent bots, providing a challenging and engaging experience.

• **Pathfinding Implementation**: Utilize the A* algorithm for bot pathfinding, allowing bots to dynamically adapt their pursuit strategy based on the player's movements.

• **Game Visualization**: Implement a visually appealing grid-based maze with distinct visual elements for walls, bots, players, and the destination tile to enhance user experience.

• **Real-Time Interaction**: Enable smooth, real-time movement for both the player and bots, ensuring seamless gameplay and responsive controls.
• **Replayability**: Introduce randomness in destination tile placement to make each game session unique, encouraging players to replay and refine their strategies.
• **User Interface Design**: Design an intuitive control system that allows players to easily navigate the maze and interact with the game without a steep learning curve.
• **Collision and Win Detection**: Implement mechanisms to detect collisions between the player and bots or the player reaching the destination tile, ensuring accurate game state management.
• **Performance Optimization**: Optimize game logic and algorithms to handle realtime decision-making and pathfinding for bots efficiently, even in a large 50x50 grid.
• **Testing and Validation**: Provide testing mechanisms to validate the correctness of bot pathfinding, collision detection, and overall game mechanics to ensure a smooth and error-free experience.

1.3 SCOPE

The scope of this project is defined as follows:

• **Grid Representation**: Implement a 50x50 grid-based maze to serve as the game arena, clearly visualizing walls, the player, bots, and the destination tile.

• **Pathfinding Algorithm**: Utilize the A* algorithm to enable bots to dynamically calculate the optimal path toward the player, ensuring intelligent and strategic bot behavior.

• **Winning and Losing Conditions**: Develop logic to detect when the player reaches the destination tile (win) or when the bots capture the player (lose), ensuring proper game state transitions.

• **User Interaction**: Enable players to navigate the maze using intuitive keyboard controls, providing smooth and responsive movement in cardinal and diagonal directions.

• **Randomized Challenges**: Introduce random destination tile placement for each game session, ensuring a unique and replayable experience.

• **Game Visualization**: Implement clear and distinct visual elements, such as colorcoded tiles for walls, bots, players, and the destination tile, to enhance game clarity and immersion.

• **Collision Detection**: Ensure accurate detection of collisions between the player, bots, and obstacles to maintain the integrity of the game mechanics.

• **Performance Optimization**: Optimize the game logic and pathfinding computations to handle real-time updates efficiently, even in a large 50x50 grid.

• **Testing and Validation**: Provide a testing framework to validate core mechanics such as pathfinding accuracy, collision handling, and game state transitions to ensure a smooth and error-free gameplay experience.

CHAPTER 2
EXISTING AND PROPOSED MODEL

2.1 Existing Models:

The existing methods for maze-based games often involve static gameplay environments or simplistic AI strategies, which present several limitations: • Static Gameplay: Many traditional maze games rely on fixed or non-dynamic player and bot interactions, resulting in predictable and less engaging gameplay experiences.

• **Simplistic Bot AI**: Bots in existing games often use basic algorithms, such as random or linear pathfinding, leading to limited strategic depth and challenges.

• **Limited Replayability**: Traditional games often lack randomization or dynamic elements, resulting in repetitive gameplay that diminishes long-term engagement.

• **Basic Visualization**: Existing maze games may not provide a clear or visually appealing representation of the maze, bots, or player, making it harder for users to follow the game's progress.

2.2 Proposed Model:

The proposed Maze Chase Game aims to overcome these challenges by offering a dynamic, engaging, and intelligent maze-based gaming experience. The proposed model includes the following features:

• **Dynamic Gameplay**: A module enabling real-time interaction where players navigate the maze while bots adapt their strategies dynamically based on the player's movements.

• **Intelligent Bots**: Integration of the A* algorithm for bot pathfinding, allowing bots to pursue the player intelligently and provide a challenging gameplay experience.

• **Randomized Challenges**: Each game session introduces randomized destination tile placement, ensuring unique and unpredictable gameplay.

• **Game Visualization**: A visually appealing grid-based maze with distinct visual elements for walls, bots, players, and destination tiles, offering clarity and immersion.

• **Collision Detection and Win Logic**: Mechanisms to detect player collisions with bots (resulting in a loss) or reaching the destination tile (resulting in a win).

• **Performance Optimization**: Optimization of game logic and pathfinding calculations to ensure smooth real-time updates, even in a large 50x50 grid.

2.3 COMPARISON OF EXISTING AND PROPOSED MODEL

The proposed Maze Chase Game provides significant improvements over traditional maze-based games:

• **Dynamic Gameplay**: Unlike static environments in existing models, the proposed game features real-time, adaptive bot strategies that keep gameplay engaging and unpredictable.

• **Advanced AI**: The A* algorithm enables bots to make intelligent decisions, offering a more challenging and strategic experience compared to simplistic bot behaviors in traditional games.

• **Enhanced Replayability**: Randomized destination tiles ensure that each gameplay session feels fresh and unique, addressing the repetitive nature of existing methods.

• **Improved Visualization**: The use of distinct color-coded elements for the maze, bots, and destination tile enhances clarity and user immersion, making it easier to understand game progress.

• **User-Friendly Interface**: Intuitive keyboard controls allow players to easily navigate the maze, ensuring accessibility for users of all ages.

• **Optimized Performance**: Real-time processing of bot pathfinding and game state transitions ensures smooth gameplay without lag, even in a complex 50x50 grid environment.

CHAPTER 3
SYSTEM DESIGN

3.1 Evaluation & Selection of Specifications/Features

To build an engaging and dynamic AI-based maze game, the following features and design specifications were selected:

- **Game Grid**: A 50x50 maze where each tile is 15x15 pixels.
- **Player Movement**: Controlled using WASD keys.
- **Bots with AI**: Two autonomous AI hunters (bots) using A* pathfinding algorithm.
- **Heuristic**: Diagonal distance heuristic for efficient chasing.
- **Obstacles**: Walls are strategically placed to make the maze challenging.
- **Real-Time Rendering**: Game elements (player, bots, walls, grid) are drawn using *pygame*.
- **Game Over Condition**: The player loses if caught by any bot.

The system was developed using Python and Pygame due to their support for 2D game development and flexibility in implementing pathfinding algorithms like A*.

3.2 Design Flow

The design of the system can be broken down into several components.

3.2.1 Class Diagram

Even though your code is function-based, you can derive a **conceptual class diagram** to represent game components:

- **Game**: Main control loop
- **Player**: Position, Movement
- **Bot**: Position, Pathfinding
- **Maze**: Walls, Grid
- **AStar**: Path calculation

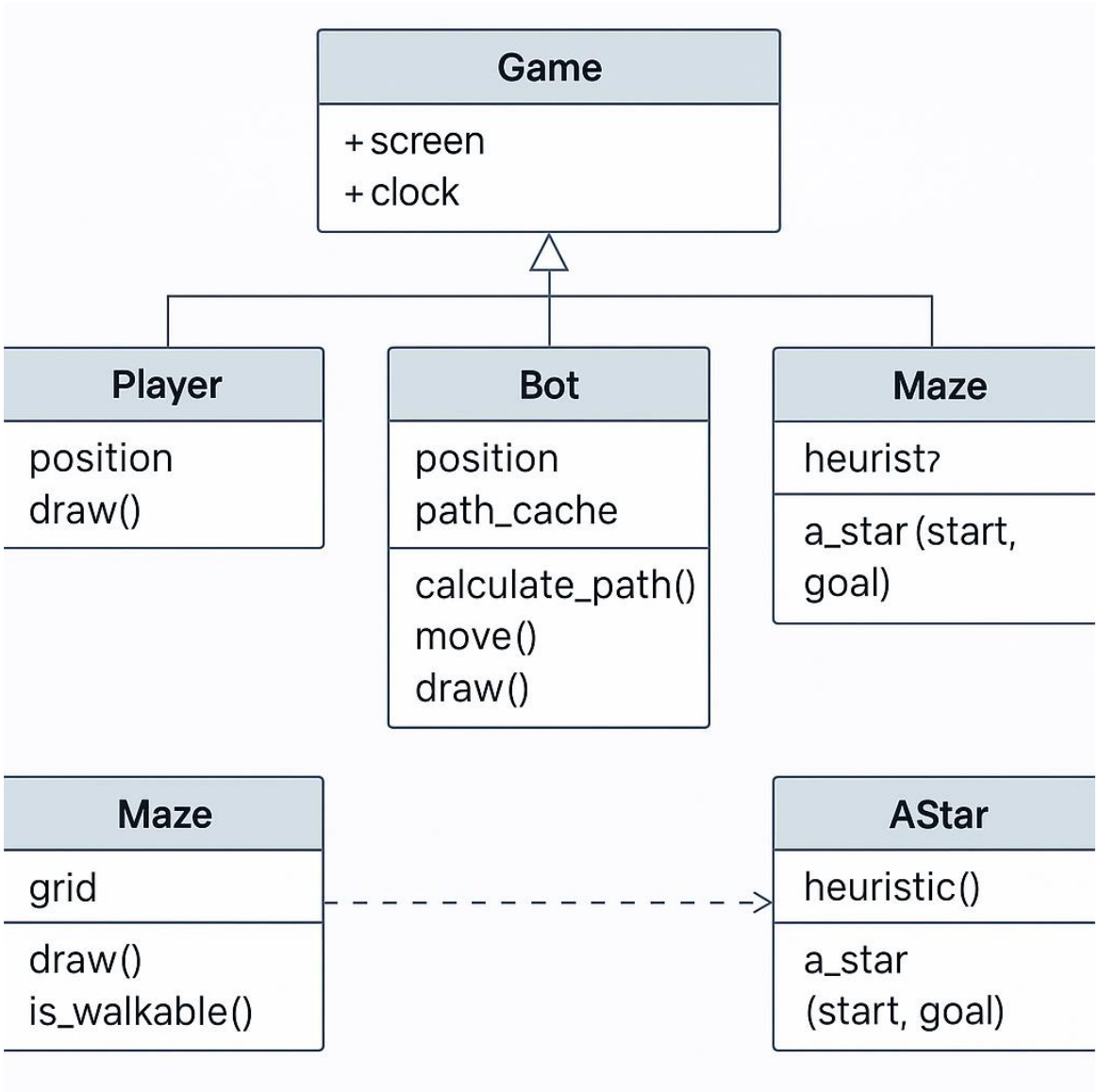

Figure 3.1 : Class Diagram

3.2.2 Flow Chart

Shows the step-by-step flow of the game:

1. Initialize game and variables
2. Draw game screen
3. Wait for player input
4. Move player and recalculate paths
5. Move bots along shortest path
6. Check for collision
7. End game if player is caught

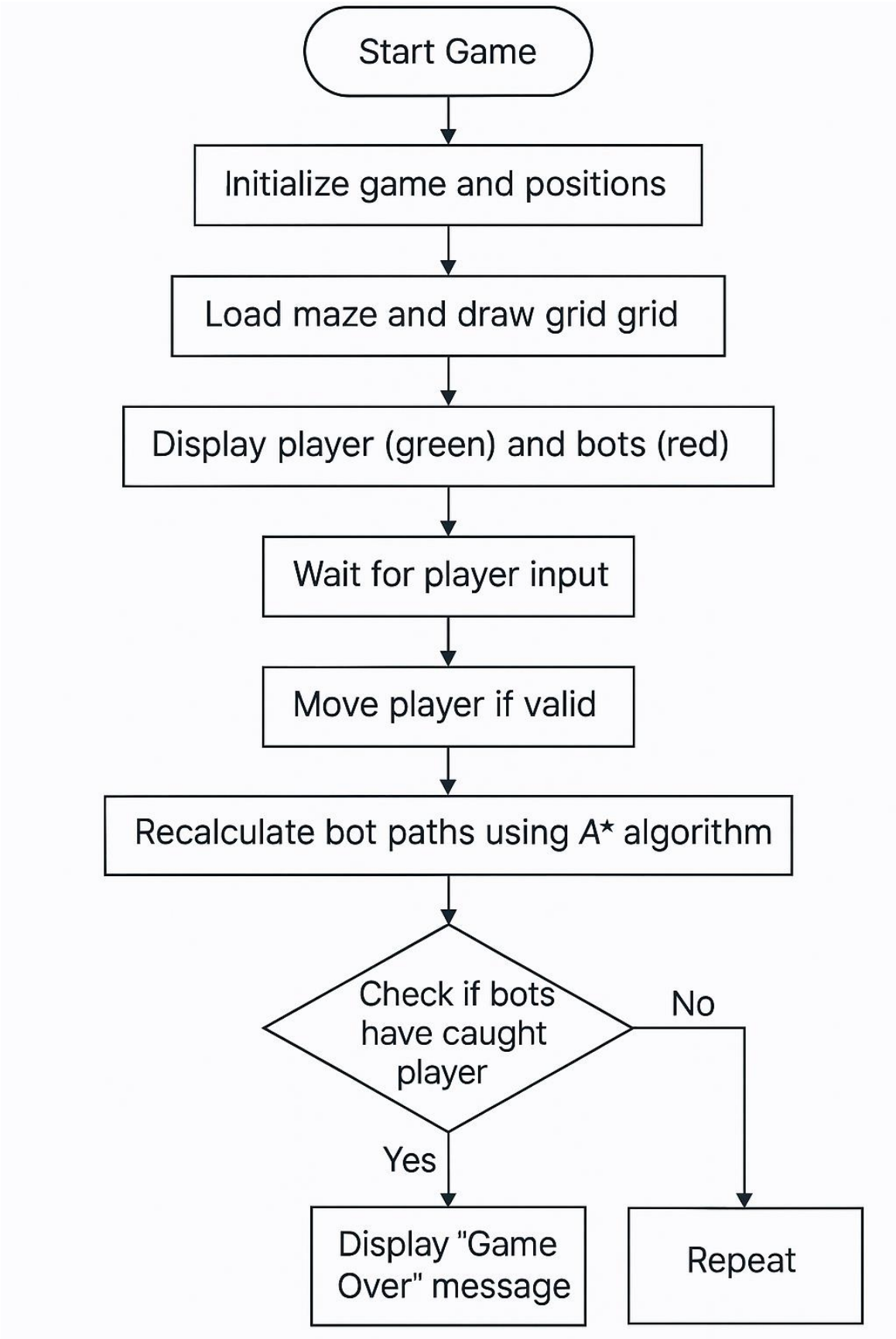

Figure 3.2 : Flowchart

3.2.3 Algorithm

A* Pathfinding Algorithm:

- **Input**: Start and goal positions
- **Output**: Optimal path avoiding obstacles
- **Steps**:
    1. Add the start node to the open list.
    2. While open list is not empty:
        - Pick the node with the lowest $f = g + h$.
        - If it is the goal, reconstruct the path.
        - Else, add its valid neighbors.
        - Update their g, h, f scores.
    3. Repeat until goal is found or no path exists.

Heuristic:

Uses diagonal distance with a slight penalty for diagonal moves to estimate cost.

CHAPTER 4
SYSTEM IMPLEMENTATION

The game is implemented using Python with the Pygame library. Key implementation highlights:

- **Grid-based Environment**: numpy is used to define the grid and wall locations.
- **Player Logic**: Controlled via keyboard with collision checks against walls.
- **Bot Logic**: Uses A* pathfinding to chase the player by recalculating the path every time the player moves.
- **Game Loop**: Handles drawing, input, and bot updates.
- **End Condition**: Game ends when either bot reaches the player.

CHAPTER 5
RESULTS & DISCUSSIONS

The system successfully demonstrates intelligent behavior by the bots in pursuing the player through an obstacle-laden environment.

- Bots dynamically update their shortest path in real-time.
- The A* algorithm proves effective even with diagonal movement and obstructions.
- Real-time response ensures the player is always under threat, increasing game tension.

5.1 Test Cases and Results

Test Case	Scenario	Expected Output	Result
1	Player moves near a wall	Wall blocks movement	Passed
2	Bot chases across maze	Path dynamically updates	Passed
3	Player caught by bot	Game ends	Passed
4	Bot path through diagonal	Shortest path chosen	Passed
5	Simultaneous bot movement	No collision between bots	Passed

TABLE 5.1 : Test Cases

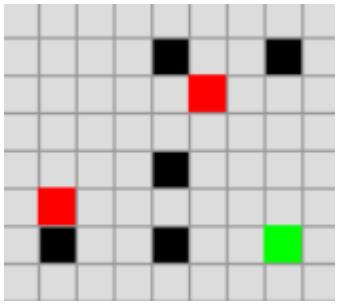

Figure 5.1 : Player and bots

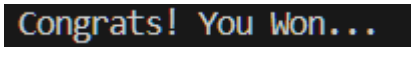

Figure 5.2 : Win UI

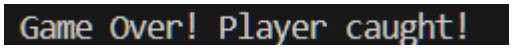

Figure 5.3 : Loss UI

CHAPTER 6
CONCLUSION & FUTURE ENHANCEMENTS

Conclusion

The Escape AI Hunters Game demonstrates real-time AI pathfinding in a dynamic maze using A* search. The project successfully integrates game logic with intelligent agents and delivers an engaging experience where bots adapt to player movement.

Future Enhancements

- Add multiple levels with increasing difficulty.
- Introduce more bots with distinct pathfinding behaviors (e.g., random walker, greedy search).
- Add power-ups or temporary walls.
- Implement scoring and UI features like menus and sound effects.

APPENDIX A

SOURCE CODE

```
import pygame
import heapq
import time
import numpy as np

# Constants
GRID_SIZE = 50
TILE_SIZE = 15
SCREEN_SIZE = GRID_SIZE * TILE_SIZE

PLAYER_COLOR = (0, 255, 0)
BOT_COLOR = (255, 0, 0)
WALL_COLOR = (0, 0, 0)
GRID_COLOR = (150, 150, 150)
BG_COLOR = (220, 220, 220)
FPS = 10

CARDINAL_COST = 1
DIAGONAL_COST = 1.414
DIAGONAL_HEURISTIC_OFFSET = DIAGONAL_COST - 2

# Directions: cardinal + diagonal moves
DIRECTIONS = [
    (-1, 0), (1, 0), (0, -1), (0, 1),
    (-1, -1), (-1, 1), (1, -1), (1, 1)
]

# Maze setup
"""
maze = np.zeros((GRID_SIZE, GRID_SIZE), dtype=int)
for i in range(15, 35):
    maze[i, 25] = 1
    """

# Mapping key events to moves (row, col)
key_map = {
    pygame.K_w: (-1, 0),
    pygame.K_s: (1, 0),
    pygame.K_a: (0, -1),
    pygame.K_d: (0, 1)
}

# Initial positions
player_pos = (0, 0)
```

```
bot1_pos = (0, 48)
bot2_pos = (48, 0)

bot1_path_cache = []
bot2_path_cache = []

def draw_grid(screen):
    """Draws the maze, player, bots, and grid lines."""
    screen.fill(BG_COLOR)

    # Draw maze walls
    for x in range(GRID_SIZE):
        for y in range(GRID_SIZE):
            if maze[x, y] == 1:
                pygame.draw.rect(screen, WALL_COLOR, (y * TILE_SIZE, x *
TILE_SIZE, TILE_SIZE, TILE_SIZE))

    # Draw grid lines
    for x in range(0, SCREEN_SIZE, TILE_SIZE):
        pygame.draw.line(screen, GRID_COLOR, (x, 0), (x, SCREEN_SIZE))
        pygame.draw.line(screen, GRID_COLOR, (0, x), (SCREEN_SIZE, x))

    # Draw player and bots
    pygame.draw.rect(screen, PLAYER_COLOR, (player_pos[1] * TILE_SIZE,
player_pos[0] * TILE_SIZE, TILE_SIZE, TILE_SIZE))
    pygame.draw.rect(screen, BOT_COLOR, (bot1_pos[1] * TILE_SIZE,
bot1_pos[0] * TILE_SIZE, TILE_SIZE, TILE_SIZE))
    pygame.draw.rect(screen, BOT_COLOR, (bot2_pos[1] * TILE_SIZE,
bot2_pos[0] * TILE_SIZE, TILE_SIZE, TILE_SIZE))

    pygame.display.flip()

def heuristic(a, b):
    """Calculate the diagonal distance heuristic for A*."""
    dx, dy = abs(a[0] - b[0]), abs(a[1] - b[1])
    return (dx + dy) + DIAGONAL_HEURISTIC_OFFSET * min(dx, dy)

def a_star(start, goal):
    """Find path from start to goal using A* algorithm."""
    open_list = [(0, start)]
    came_from = {}
    g_score = {start: 0}
    f_score = {start: heuristic(start, goal)}

    while open_list:
        _, current = heapq.heappop(open_list)
        if current == goal:
            path = []
```

```
            while current in came_from:
                path.append(current)
                current = came_from[current]
            return path[::-1]  # Return reversed path

        for dx, dy in DIRECTIONS:
            neighbor = (current[0] + dx, current[1] + dy)
            if 0 <= neighbor[0] < GRID_SIZE and 0 <= neighbor[1] <
GRID_SIZE and maze[neighbor] == 0:
                move_cost = DIAGONAL_COST if dx != 0 and dy != 0 else
CARDINAL_COST
                temp_g_score = g_score[current] + move_cost
                if neighbor not in g_score or temp_g_score <
g_score[neighbor]:
                    came_from[neighbor] = current
                    g_score[neighbor] = temp_g_score
                    f_score[neighbor] = temp_g_score + heuristic(neighbor,
goal)
                    heapq.heappush(open_list, (f_score[neighbor],
neighbor))
    return []

def move_player(event):
    """Handle player movement and update bots' paths if player moved."""
    global player_pos, bot1_path_cache, bot2_path_cache
    if event.key in key_map:
        dx, dy = key_map[event.key]
        new_pos = (player_pos[0] + dx, player_pos[1] + dy)
        if 0 <= new_pos[0] < GRID_SIZE and 0 <= new_pos[1] < GRID_SIZE and
maze[new_pos] == 0:
            player_pos = new_pos
            # Recalculate paths for bots toward the new player position
            bot1_path_cache = a_star(bot1_pos, player_pos)
            bot2_path_cache = a_star(bot2_pos, player_pos)
            return True
    return False

def move_bots():
    """Move bots along their precomputed paths."""
    global bot1_pos, bot2_pos

    if bot1_path_cache:
        next_step = bot1_path_cache.pop(0)
        if maze[next_step] == 0 and next_step != bot2_pos:
            bot1_pos = next_step

    if bot2_path_cache:
        next_step = bot2_path_cache.pop(0)
```

```
        if maze[next_step] == 0 and next_step != bot1_pos:
            bot2_pos = next_step

    # Check for collision with player
    if bot1_pos == player_pos or bot2_pos == player_pos:
        time.sleep(1)
        print("Game Over! Player caught!")
        pygame.quit()
        exit()

def main():
    global player_pos, bot1_pos, bot2_pos
    pygame.init()
    screen = pygame.display.set_mode((SCREEN_SIZE, SCREEN_SIZE))
    pygame.display.set_caption("Maze Chase Game")
    clock = pygame.time.Clock()

    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            elif event.type == pygame.KEYDOWN:
                if move_player(event):
                    move_bots()

        draw_grid(screen)
        clock.tick(FPS)

    pygame.quit()

if __name__ == '__main__':
    main()
```

REFERENCES

1. Russell, S. J., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach*.
2. Pygame Documentation – https://www.pygame.org/docs/
3. Amit Patel's Guide to A* Pathfinding – https://www.redblobgames.com/pathfinding/a-star/introduction.html