CSE 4110: Artificial Intelligence Laboratory

SOS paper-and-pencil AI Game

By

Sk. Azraf Sami

Roll: 1907115

Atiqul Islam Atik

Roll: 1907107

Submitted To:

Md. Shahidul Salim

Lecturer

Department of Computer Science and Engineering

Khulna University of Engineering & Technology

Most. Kaniz Fatema Isha

Lecturer

Department of Computer Science and Engineering

Khulna University of Engineering & Technology

Department of Computer Science and Engineering

Khulna University of Engineering & Technology

Khulna 9203, Bangladesh

September, 2024

Acknowledgement

All the praise to the almighty Allah, whose blessing and mercy succeeded me to complete this course work fairly. I gratefully acknowledge the valuable suggestions, advice and sincere cooperation of my course teachers, Md. Shahidul Salim & Most. Kaniz Fatema Isha.

Contents

CHAPTER I

Introduction

1.1 Introduction

The SOS game is a simple two-player game that is typically played on a grid. The goal is to form the sequence "SOS" in any row, column, or diagonal.

- Two player game
- Opponent is AI moved
- Two player options available

1.2 How To Play

Players: The game is for two players. They take turns writing either an "S" or an "O" in an empty cell.
Winning: A player wins by forming the sequence "SOS" horizontally, vertically, or diagonally. Each time a player forms "SOS", they score a point and continue their turn.
Game End: The game ends when all the cells are filled. The player with the most "SOS" sequences wins.

1.3 Objectives

1. **Develop a Multi-Level Difficulty System**:
    - Implement four distinct AI difficulty modes: Easy (Fuzzy Logic), Medium (Genetic Algorithm), Hard (A* Algorithm), and Very Hard (Minimax with Alpha-Beta Pruning).
2. **Implement AI vs. Human and Human vs. Human Modes**:

- Design and integrate two gameplay modes: AI vs. Human, where the AI adapts based on the selected difficulty, and Human vs. Human, where two players can compete.

3. **Optimize AI Strategies for Efficient Decision-Making**:
    - Enhance the AI's decision-making process using techniques like alpha-beta pruning to reduce computational overhead, especially in the Very Hard mode.
4. **Integrate Heuristic and Genetic Algorithms for Medium Difficulty**:
    - Apply genetic algorithms and heuristic-based strategies to create a challenging yet manageable AI opponent in the Medium difficulty mode.
5. **Enhance Gameplay Experience through Thorough Testing**:
    - Conduct comprehensive testing of AI and human interactions to ensure balanced gameplay and refine AI responses based on player actions.
6. **Create a Responsive User Interface**:
    - Develop a user-friendly interface that adapts to the selected gameplay mode, displaying relevant options and controls for both AI vs. Human and Human vs. Human modes.

CHAPTER II

Theory

2.1 Overview

The AI for the Paper-Pencil SOS game is designed to provide a challenging and adaptive gameplay experience across multiple difficulty levels. The core of the AI's decision-making process is based on the Minimax algorithm, which evaluates all possible moves to determine the most optimal outcome. To enhance efficiency, Alpha-Beta Pruning is applied, reducing the number of nodes that need to be evaluated and thus optimizing the decision-making process. Additionally, Generative AI techniques are integrated to create and refine strategies dynamically, allowing the AI to adapt to different game scenarios and improve over time. These combined methodologies ensure a robust and responsive AI that can challenge players at varying levels of difficulty.

2.2 Fuzzy Logic

Fuzzy Logic is a reasoning method that deals with approximate or uncertain information, enabling decisions based on imprecise data. This approach is particularly useful in evaluating moves within the game, where exact calculations may not always be possible.

- **Objective**: Enhance AI decision-making by evaluating moves with fuzzy logic.

2.2.1 Components

- **Fuzzification**:
  - **Purpose**: Convert game scores into fuzzy categories (e.g., bad, average, good).
  - **Function**: `fuzzify_score(score)`
  - **Example**: Score 0 → {'bad': 1, 'average': 0, 'good': 0}.

- **Defuzzification**:
  - **Purpose**: Translate fuzzy values into a numerical score for decision-making.
  - **Function**: `defuzzify(fuzzy_scores)`
- **Move Evaluation**:
  - **Purpose**: Simulate and assess moves using fuzzy logic to determine their potential.
  - **Function**: `evaluate_move_fuzzy(board, row, col, char)`
- **Finding and Making Moves**:
  - **Purpose**: Identify the best move based on fuzzy evaluation.
  - **Function**: `find_best_move(board)`
  - **AI Decision-Making**: Executes the best move to enhance gameplay.

2.3 Genetic Algorithm

The Genetic Algorithm (GA) for the SOS game involves using evolutionary techniques to optimize strategies or solutions for the game. The main purpose is to evolve solutions over generations using the principles of natural selection.

- **Purpose**: Evolve solutions over generations using principles of natural selection.

2.3.1 Components

- **Population**:
  - **Definition**: A set of possible solutions.
- **Fitness Function**:
  - **Purpose**: Evaluates the quality of solutions.
- **Selection**:
  - **Purpose**: Chooses the best solutions for reproduction.

- **Crossover**:
  - **Purpose**: Combines solutions to create new ones.
- **Mutation**:
  - **Purpose**: Introduces random changes to solutions.

2.3.2 GA Components in SOS

1. **Population Initialization**:
   - **Representation**: Encode strategies or moves as chromosomes (sequences of moves).
2. **Fitness Function**:
   - **Evaluation**: Score strategies based on game outcomes (win/loss, SOS formations).
3. **Selection Process**:
   - **Method**: Select strategies with higher fitness for breeding (roulette wheel selection).
4. **Crossover and Mutation**:
   - **Crossover**: Combine strategies from two-parent solutions.
   - **Mutation**: Introduce random modifications to strategies.

2.3.3 Function Descriptions

- **evaluate_board(board)**: Calculates the board's score based on 'SOS' and 'SSO' patterns.
- **initialize_population(board, population_size=10)**: Generates initial random moves and characters for the genetic algorithm.
- **evaluate_fitness(board, individual)**: Measures an individual's fitness by simulating their move on the board.

- **select_parents(population, fitnesses)**: Chooses two parents based on their fitness scores.
- **crossover(parent1, parent2)**: Combines traits from two parents to create a new individual.
- **mutate(individual, mutation_rate=0.1)**: Applies random changes to an individual.
- **genetic_algorithm(board,generations=10,population_size=10, mutation_rate=0.1)**: Evolves the population to find the best move and character.

2.4 A* Algorithm

The A* algorithm is a pathfinding and search algorithm used to find the optimal move in a game by evaluating possible future states based on a heuristic function.

- **Purpose**: To determine the best move for AI in a turn-based game by evaluating potential board states.
- **Heuristic Function**: Utilizes `evaluate_board` to assess board configurations and prioritize promising moves.

2.4.1 Evaluation Function

- **Function**: `evaluate_board(board)`
- **Purpose**: Computes a score for the current board state based on patterns of 'S' and 'O'.
- **Details**: Checks for sequences of 'S' and 'O' in rows, columns, and diagonals to assign scores.

2.4.2 Heuristic Function

- **Function**: `heuristic(board)`
- **Purpose**: Calls `evaluate_board` to return the score of the board state.

2.4.3 Move Generation

- **Function**: `get_possible_moves(board)`
- **Purpose**: Identifies all possible moves (empty cells) on the board.

2.4.4 A* Search

- **Function**: `a_star(board, max_depth)`
- **Purpose**: Searches for the optimal move by simulating moves up to a specified depth.
- **Details**:
  - Uses a priority queue (heap) to explore moves.
  - Evaluates moves by placing 'S' or 'O' and updating scores.
  - Keeps track of the best move and score.

2.5 Minimax Algorithm

The Minimax algorithm is a decision-making algorithm commonly used in game theory and artificial intelligence to determine the optimal move by simulating all possible actions and responses.

- **Purpose**: AI's decision-making process.
- **Minimax**: Maximizes the AI's chances while minimizing the opponent's advantage.
- **Alpha-Beta Pruning**: Optimizes performance by reducing unnecessary calculations.

2.5.1 Key Components

- **Board Evaluation**:
  - **Function**: `evaluate_board(board)`
  - **Purpose**: Scores based on SOS formations (horizontal, vertical, diagonal).
- **Move Availability**:
  - **Function**: `is_moves_left(board)`

  - **Purpose**: Checks if any moves are left on the board.

- **Minimax Execution**:
  - **Function**: `mini_max(board,depth,is_max,alpha,beta,max_depth)`
  - **Maximizing Player**: AI's perspective.
  - **Minimizing Player**: Opponent's perspective.
  - **Alpha-Beta Pruning**: Optimizes decision-making by pruning branches of the game tree.

CHAPTER III

Functionalities

3.1 Functionalities

The SOS game involves a grid-based board game with player and AI interactions, win conditions, and various UI elements. Below is a detailed overview of the game's functionality:

3.1.1 Game Initialization

- **Window and Board Initialization:** Creates the game window using Tkinter and initializes the SOS grid.
- **Player and AI Setup:** Handles player and AI initialization, including setting up the SOS grid and tracking player moves.
- **Game Sounds:** Integrates game sounds for notifications such as win/lose using Pygame or other libraries.

3.1.2 Board Setup and Management

- **Board Initialization:** Initializes and displays the SOS game board with a grid layout.
- **Drawing Symbols:** Handles the drawing and updating of game symbols (S, O) on the board.
- **Move Management:** Manages player and AI moves, ensuring valid placements and updating the board accordingly.

3.1.3 Player and AI Interaction

- **Player Moves:** Allows players to make moves by clicking on the board or using other input methods.

- **AI Moves:** Handles AI moves based on difficulty levels, such as random or strategic moves.
- **Feedback:** Provides feedback on player actions, such as highlighting selected cells or indicating valid/invalid moves.

3.1.4 Game Logic and Win Conditions

- **Win Conditions:** Implements the rules for checking win conditions (e.g., three in a row for SOS).
- **Game State Management:** Manages game state transitions, including detecting wins, losses, or draws.
- **Notifications:** Handles notifications and alerts for game outcomes and player turns.

3.1.5 User Interface (UI) and Interaction

- **Interface Design:** Provides a user-friendly interface for starting new games, selecting difficulty levels, and viewing game results.
- **Controls:** Integrates buttons and controls for game actions, such as resetting the board or changing settings.
- **Information Display:** Displays game information and instructions, such as turn indicators and scoreboards.

3.1.6 Movement and Placement Rules

- **Valid Moves:** Ensures that players can only place symbols in empty cells and follow the rules for SOS.
- **Move Validation:** Validates moves to prevent players from placing symbols outside the board or in occupied cells.
- **State Updates:** Manages game state updates in response to valid moves and checks for win conditions after each move.

3.1.7 Tkinter and Pygame Integration

- **Initialization:** Initializes Pygame for sound and animation, and creates a Tkinter window for the game interface.
- **Audio Management:** Uses Pygame to play background music and button click sounds.

3.1.8 Difficulty Selection

- **Difficulty Options:** Provides options to select game difficulty and loads the corresponding AI algorithm.

3.1.9 Multiplayer Mode

- **Player Setup:** Prompts for player names and opens a new window for multiplayer gameplay.
- **Sound and Visuals:** Integrates sound effects and background images for a more engaging experience.

3.1.10 Story and Rules Display

- **Story and Rules:** Shows the game's backstory and rules in separate Tkinter windows.

3.1.11 Game Board Setup

- **Board Design:** Initializes a game board with buttons for gameplay.
- **Visual Effects:** Loads animated GIFs for visual effects on the board.

3.1.12 AI Move Handling

- **AI Strategies:** Manages AI moves, checking for the best move based on the selected difficulty level.

CHAPTER IV

UI Design

4.1 Game UI

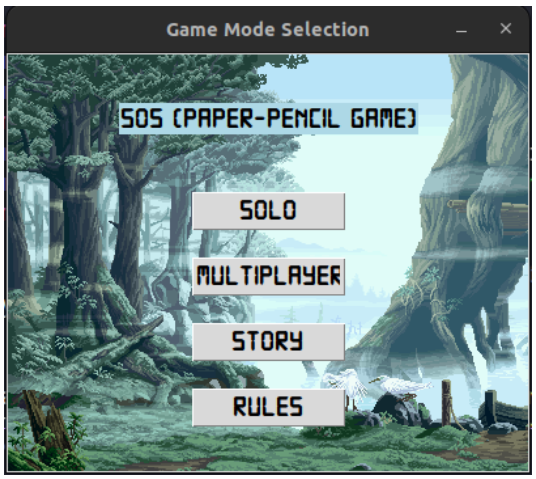

Figure 4.1: Home Menu

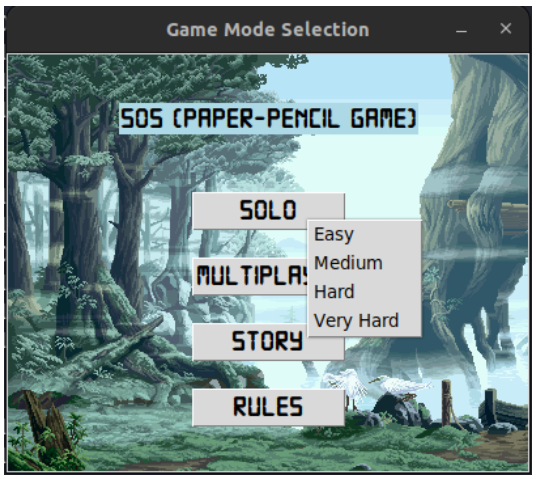

Figure 4.2: Mode Selection

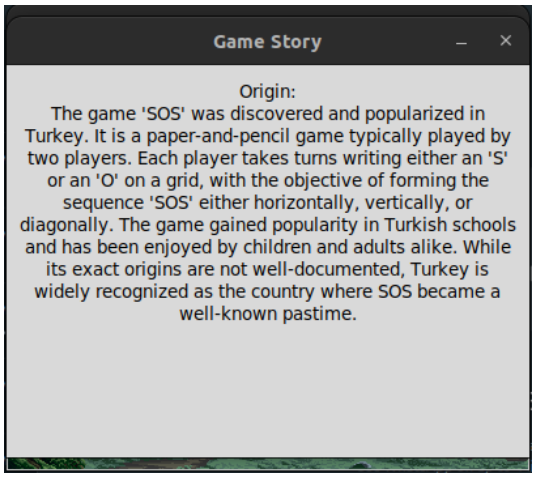

Figure 4.3: Game Story

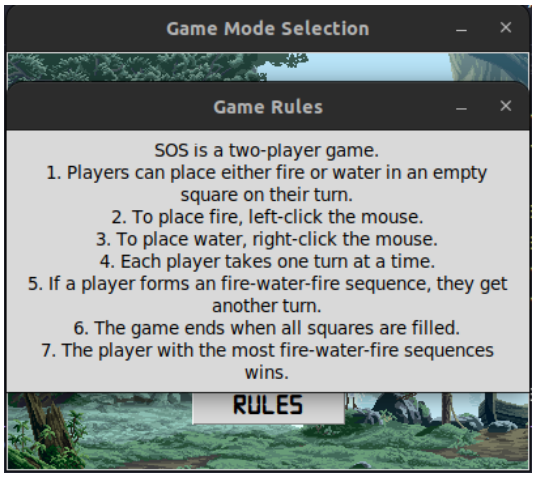

Figure 4.4: Rules

Figure 4.5: Player's Name

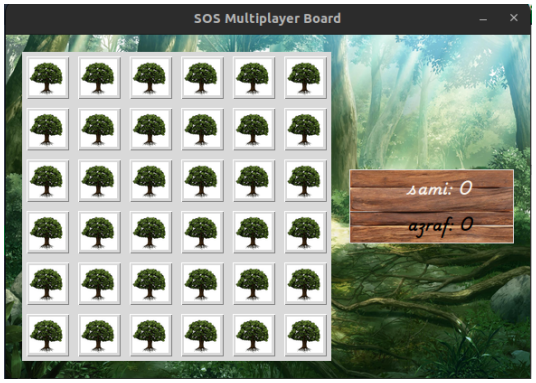

Figure 4.6: Multiplayer Board

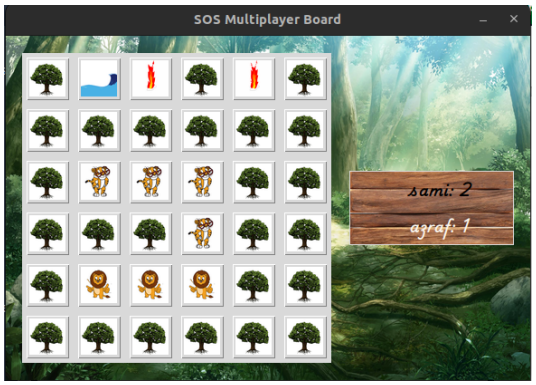

Figure 4.7: Interaction on Multiplayer Board

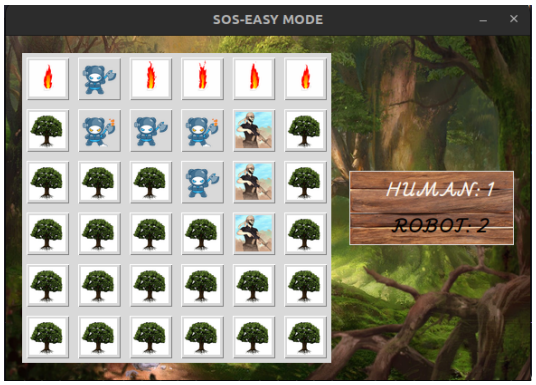

Figure 4.8: Easy Mode using Fuzzy Logic

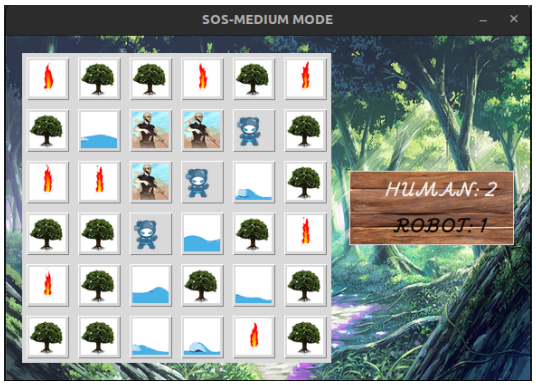

Figure 4.9: Medium Mode using Genetic Algo.

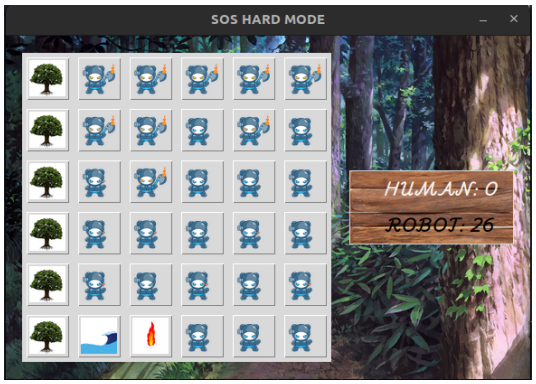

Figure 4.10: Hard Mode using A*

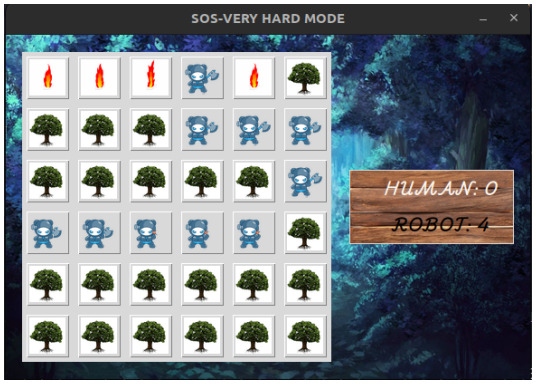

Figure 4.11: Very Hard Mode using Min-Max

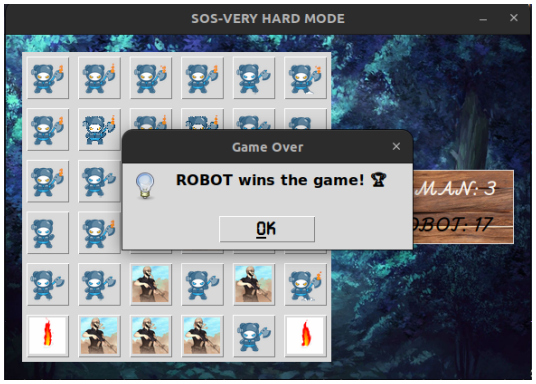

Figure 4.12: Result Widow Box

4.2 Debug Console

```
Selected difficulty: Medium
1 S at (1, 2)
(0,0): Empty | (0,1): Empty | (0,2): Empty | (0,3): Empty | (0,4): Empty | (0,5): Empty |
(1,0): Empty | (1,1): Empty | (1,2): S | (1,3): Empty | (1,4): Empty | (1,5): Empty |
(2,0): Empty | (2,1): Empty | (2,2): Empty | (2,3): Empty | (2,4): Empty | (2,5): Empty |
(3,0): Empty | (3,1): Empty | (3,2): Empty | (3,3): Empty | (3,4): Empty | (3,5): Empty |
(4,0): Empty | (4,1): Empty | (4,2): Empty | (4,3): Empty | (4,4): Empty | (4,5): Empty |
(5,0): Empty | (5,1): Empty | (5,2): Empty | (5,3): Empty | (5,4): Empty | (5,5): Empty |

2 S at (4, 0)
(0,0): Empty | (0,1): Empty | (0,2): Empty | (0,3): Empty | (0,4): Empty | (0,5): Empty |
(1,0): Empty | (1,1): Empty | (1,2): S | (1,3): Empty | (1,4): Empty | (1,5): Empty |
(2,0): Empty | (2,1): Empty | (2,2): Empty | (2,3): Empty | (2,4): Empty | (2,5): Empty |
(3,0): Empty | (3,1): Empty | (3,2): Empty | (3,3): Empty | (3,4): Empty | (3,5): Empty |
(4,0): S | (4,1): Empty | (4,2): Empty | (4,3): Empty | (4,4): Empty | (4,5): Empty |
(5,0): Empty | (5,1): Empty | (5,2): Empty | (5,3): Empty | (5,4): Empty | (5,5): Empty |
```

Figure 4.13: Status of Board at Each Move

4.3 UI Components and Functionality

4.3.1 Tooltips

- **Function:** Shows additional information when the user hovers over UI elements.
- **Methods:**
  - `show_tooltip`: Displays the tooltip with the relevant information.
  - `hide_tooltip`: Hides the tooltip when the user moves away from the element.
  - `bind_tooltip`: Binds the tooltip to a specific UI element.

4.3.2 Scoreboard

- **Function:** Tracks and displays the scores of players throughout the game.
- **Methods:**

- `update_scoreboard`: Updates the scoreboard with current scores.
- `increment_score`: Increments the score for a specific player.

4.3.3 Animations

- **Function:** Provides visual feedback for player actions and game events.
- **Method:**
  - `update_button_image`: Updates the image or visual effect of a button to reflect user interaction.

4.3.4 Game Mechanics

- **SOS Detection:** Identifies and validates 'SOS' patterns on the board.
- **Method:**
  - `check_sos`: Checks the board for any 'SOS' patterns and determines if a player has won.

4.3.5 Event Handling

- **Function:** Processes and manages user and AI moves.
- **Methods:**
  - `handle_click`: Processes the user's click and updates the game state.
  - `handle_click_ai`: Manages the AI's move and updates the game state accordingly.

4.3.6 Game End

- **Function:** Determines when the game has ended and displays the result.
- **Methods:**
  - `check_game_end`: Checks the game state to determine if the game is over.
  - `show_winner_message`: Displays a message indicating the winner or the result of the game.

4.3.7 Button Management

- **Function:** Controls the interactivity of buttons during gameplay.
- **Methods:**
  - `disable_all_buttons`: Disables all buttons to prevent further interaction.
  - `enable_all_buttons`: Enables all buttons to allow player interaction.

4.3.8 Board State Debugging

- **Function:** Provides a method for printing and reviewing the current status of the board for debugging purposes.
- **Method:**
  - `print_board`: Prints the current state of the board to the console or a log file.

CHAPTER V
Discussion and Conclusion

5.1 Discussion

The SOS game is a strategic two-player board game designed with an emphasis on intuitive user interactions and dynamic gameplay. It operates on a grid where players place 'S' and 'O' symbols, aiming to create the 'SOS' pattern. Key functionalities include real-time move validation, win condition detection, and interactive user feedback through tooltips, scoreboards, and animations. The game integrates Tkinter for the graphical interface and Pygame for sound effects, ensuring a seamless and engaging user experience. With support for both human and AI opponents, the game adapts to different difficulty levels, providing varied and challenging gameplay. This design approach enhances both the visual and functional aspects, making the game accessible and enjoyable for players of all levels.

5.2 Conclusion

In conclusion, the SOS game project effectively combines interactive design and strategic gameplay to deliver an engaging experience for users. Through its well-defined functionalities—ranging from real-time move validation and dynamic score tracking to intuitive user interfaces and adaptive AI difficulty levels—the game provides a robust platform for both casual and competitive play. The integration of Tkinter and Pygame ensures a polished graphical and auditory experience, while the modular approach to game mechanics and user interaction fosters ease of maintenance and scalability. This project demonstrates how thoughtful design and technology integration can create a compelling game that is both enjoyable and accessible, reflecting a successful implementation of game development principles.

5.3 Future Work

- **Enhanced AI Algorithms:** Develop and integrate more advanced AI strategies for higher difficulty levels, including heuristic-based and machine learning approaches.
- **Multiplayer Mode Expansion:** Introduce online multiplayer functionality to allow players to compete with others remotely.
- **Game Customization:** Add options for users to customize game settings, such as board size, game symbols, and themes.
- **Improved Graphics and Animations:** Enhance visual effects and animations for a more immersive gaming experience, including smoother transitions and more dynamic board interactions.
- **Advanced User Interface:** Implement a more sophisticated UI with additional features such as player profiles, leaderboards, and in-game tutorials.
- **Mobile Compatibility:** Develop a mobile version of the game to make it accessible on smartphones and tablets.
- **Game Analytics and Feedback:** Incorporate analytics to track player performance and gather feedback to continually improve gameplay and user experience.
- **Accessibility Features:** Integrate accessibility options to make the game more inclusive, such as support for screen readers and customizable controls.
- **Story Mode or Campaign:** Introduce a single-player story mode or campaign with progressive levels and challenges to enhance replayability.
- **Sound and Music Enhancements:** Expand the audio experience with more varied sound effects and background music options to enrich the overall atmosphere of the game.