MKSSS's Cummins COLLEGE OF ENGINEERING FOR WOMEN

AI MINI PROJECT REPORT

ON

" CHECKERS GAME "

Submitted By

UIT2021851- Tarushi Nimje
UIT2021852- Mahima Patil
UIT2021867- Rutuja Waghmode

DEPARTMENT OF INFORMATION TECHNOLOGY
MKSSS's CUMMINS COLLEGE OF ENGINEERING FOR WOMEN,
PUNE
SAVITRIBAI PHULE PUNE UNIVERSITY
2022 – 2023

Contents of mini project

ABSTRACT

This report presents the implementation and analysis of a checkers game utilizing the Minimax algorithm for artificial intelligence decision-making. The project aims to develop an intelligent agent capable of strategic gameplay, employing the Minimax algorithm to make optimal moves in a two-player checkers game. The implementation is done in Python, integrating the Pygame library for graphical user interface elements.

The report outlines the key components of the project, including the game structure, the Minimax algorithm for decision-making, and the graphical representation of the checkers board. The algorithm's functionality is examined in detail, showcasing its role in determining optimal moves for the AI player.

The project's success is evaluated through gameplay outcomes and an analysis of the AI's decision-making process. The report concludes with reflections on the project's achievements, challenges faced, and potential areas for future enhancements.

INTRODUCTION

In the realm of artificial intelligence and strategic board games, the implementation of intelligent agents capable of making optimal decisions presents a fascinating and challenging endeavor. This report documents the development and analysis of a checkers game with an integrated artificial intelligence component, employing the Minimax algorithm.

1. Background:

Begin by introducing the background of the project. Discuss the significance of artificial intelligence in board games, emphasizing the strategic complexity inherent in games like checkers. Highlight the relevance of the Minimax algorithm as a decision-making tool for optimizing gameplay.

2. Motivation:

Explain the motivation behind choosing checkers as the focus of this project. Discuss the challenges associated with developing an intelligent agent capable of competing with human players in a two-player, turn-based board game.

3. Objectives:

Clearly outline the objectives of the project. Specify the goals, such as implementing the checkers game, integrating the Minimax algorithm, and evaluating the AI's performance in strategic decision-making.

4. Scope:

Define the scope of the project by outlining its limitations and constraints. Discuss any specific aspects of the checkers game or the Minimax algorithm that you decided to focus on or exclude from the scope of this project.

5. Structure of the Report:

Provide a brief overview of how the report is organized. Mention the key sections that readers can expect, such as the methodology, implementation details, results, and conclusions.

PROBLEM STATEMENT

Design and implement an intelligent checkers-playing agent using the Minimax algorithm in Python. This project focuses on adapting Minimax to the complexities of checkers gameplay, incorporating a user-friendly graphical interface, and providing comprehensive documentation for transparency and future improvements.

OBJECTIVES

The overarching objectives of the checkers AI project are designed to address the identified challenges and contribute to the advancement of intelligent agents in board games, with a focus on the game of checkers. The key objectives include:

1. Implementing a Functional Checkers Game:

Develop a fully functional checkers game with a user-friendly graphical interface. Ensure that the game adheres to the standard rules of checkers and provides an engaging platform for human and computer players.

2. Integrating the Minimax Algorithm:

Integrate the Minimax algorithm into the checkers game to enhance the strategic decision-making capabilities of the computer player. Implement the algorithm with careful consideration of the specific complexities associated with checkers gameplay.

3. Evaluating the AI's Performance:

Conduct thorough evaluations of the AI player's performance. Assess its ability to make strategic decisions, predict opponent moves, and optimize gameplay. Use a variety of test scenarios to validate the effectiveness of the Minimax algorithm in enhancing the AI's playing capabilities.

4. Enhancing User Experience:

Prioritize the user experience by implementing user-friendly controls, clear graphical representations, and interactive features. Ensure that the game provides an enjoyable experience for both human and computer players, striking a balance between challenge and engagement.

5. Documentation and Analysis:

Document the implementation details, algorithms used, and any challenges faced during the development process. Perform a comprehensive analysis of the project outcomes, including insights into the strengths and limitations of the implemented checkers AI.

6. Providing a Platform for Future Work:

Create a foundation for future work in the field of intelligent agents in board games. Develop the project in a modular and extensible manner, facilitating potential enhancements, research, or educational use.

PEAS

The PEAS (Performance measure, Environment, Actuators, Sensors) description outlines the key components of an intelligent system, in this case, a checkers game played by a computer agent.

Performance Measure:

Objective: The performance measure defines the criteria for evaluating the success of the checkers game.

- Accuracy of Moves: The system's ability to accurately execute legal moves based on the rules of checkers.
- Competitive Gameplay: Success in competitive play against human or AI opponents, gauged by win/loss ratios and strategic decision-making.
- User Satisfaction: Incorporating user feedback and engagement metrics to ensure a satisfying gaming experience.

Environment:

- Objective: Describes the external factors that the intelligent system interacts with during the checkers game.
    - Game Board: The 8x8 square grid representing the checkers board where the game is played.
    - Opponent's Moves: The dynamic and changing nature of the opponent's moves and strategies in response to the computer's actions.
    - User Interface: The graphical representation of the game, including the visual elements, controls, and feedback mechanisms.

Actuators:

- Objective: Actuators are the mechanisms through which the intelligent system influences the environment.
    - Piece Movement: The system's ability to move checkers pieces on the board in accordance with the rules.
    - Capturing Mechanism: Executing captures by jumping over opponent pieces when applicable.
    - User Interface Interactions: Responding to user inputs, including mouse or touchscreen actions for move selection.

Sensors:

- Objective: Sensors are the mechanisms through which the intelligent system perceives or gathers information about the environment.

- Board State Detection: Sensors that perceive the current state of the checkers board, including the positions of all pieces.
- User Input Recognition: Sensors that interpret user inputs, such as selecting a piece to move or indicating a move destination.
- Opponent's Moves Recognition: Detecting and interpreting the moves made by the opponent, either human or computer-controlled.

ENVIRONMENT

The environment of a checkers game refers to the external factors and conditions that influence and interact with the game. Here are the detailed components of the environment in a checkers game:

1. Game Board:

- Description: The primary space where the checkers game unfolds.
- Details:
    - 8x8 Grid: The board consists of 64 squares arranged in an 8x8 grid, each square alternating between dark and light colors.
    - Starting Positions: Three rows on each side hold the starting positions for each player's 12 pieces.
    - Coordinate System: A coordinate system is used to identify each square on the board uniquely.

2. Pieces:

- Description: The individual components that players move on the board.
- Details:
    - Two Sets of Pieces: Typically, there are two sets of pieces, each with 12 distinctively colored markers.
    - Regular Pieces (Men): Initially move diagonally forward and can be kinged upon reaching the opponent's back row.
    - Kinged Pieces: Can move both diagonally forward and backward.

3. Opponent's Moves:

- Description: The dynamic and changing nature of the opponent's actions.
- Details:
    - Dynamic Strategies: The opponent, whether human or computer-controlled, adapts its moves based on the player's actions and the evolving state of the game.
    - Legal Move Generation: The opponent identifies legal moves, captures, and strategic positioning to challenge the player.

4. User Interface:

- Description: The graphical representation of the game that allows user interaction.
- Details:
    - Visual Elements: Graphics representing the board, pieces, and other relevant game information.

    - Interactive Controls: Buttons, touchpoints, or mouse interactions enabling the player to make moves.
    - Feedback Mechanisms: Visual and audio cues to provide feedback on moves, captures, and game state.

5. Rules and Constraints:

- Description: The predefined rules that govern the checkers game.
- Details:
    - Movement Rules: Rules governing how pieces can move, capture, and be kinged.
    - Victory Conditions: Conditions determining when a player wins, such as capturing all opponent pieces or placing them in a position with no legal moves.
    - Stalemate Conditions: Conditions leading to a draw if neither player can make a legal move.

6. Dynamic State:

- Description: The constantly evolving state of the game.
- Details:
    - Piece Positions: The current positions of all pieces on the board.
    - Kinged Pieces: Tracking which pieces are kinged.
    - Captures and Jumps: Recording sequences of captures and jumps during the game.

SOFTWARE AND HARDWARE REQUIREMENTS

The hardware and software requirements for the provided Checkers game code depend on the specific dependencies and technologies used in the implementation. However, I can provide general recommendations based on common Python game development practices:

Hardware Requirements:

Processor (CPU): A multi-core processor (e.g., Intel Core i5 or equivalent) is recommended for smoother execution.
Memory (RAM): A minimum of 4GB RAM is advisable. More RAM may be beneficial for larger games.
Graphics Card: A dedicated graphics card is not strictly necessary for this 2D game, but having one can improve graphical performance, especially if using Pygame.
Storage: The game itself doesn't require significant storage space. A few hundred megabytes should be more than enough.

Software Requirements:

Operating System: The code should work on various operating systems, including Windows, macOS, and Linux.
Python: The code is written in Python, so you need to have Python installed. The recommended version is Python 3.6 or later.
Pygame Library: As the game uses Pygame for graphical rendering, you need to

install the Pygame library.

pygame-image Library: If you're using images in your game (e.g., for the crown

icon), ensure you have the required image libraries installed.

Development Environment:

Code Editor or IDE: Use a code editor or integrated development environment (IDE) of your choice. Popular choices include Visual Studio Code, PyCharm, or IDLE.

Git (Optional): If you plan to collaborate on the code or use version control, having Git installed is beneficial.

ALGORITHM USED

The Checkers game employs the Minimax algorithm to power its artificial intelligence, creating a strategic opponent for players. Implemented in Python using the Pygame library, the algorithm explores the game tree to determine optimal moves for the AI player.

Core Minimax Function

In the `algorithm.py` file, the `minimax` function is central to the algorithm. It recursively explores the game tree, considering the current game state (`position`), search depth, and the player's turn. Key features include:

- Termination Conditions: The function terminates when the search depth is 0 or a terminal state (a player wins).
- Maximizing and Minimizing Players: For the maximizing player's turn, the algorithm maximizes the evaluation, and for the minimizing player, it minimizes the evaluation.
- Move Selection: The best move is determined based on the maximum or minimum evaluation, depending on the player's turn.

Supporting Functions

Two crucial supporting functions complement the Minimax algorithm:

Move Simulation (`simulate_move`): Simulates a move on the board, updating the game state without affecting the actual game.
Move Generation (`get_all_moves`): Generates all possible moves for a given color on the current game board.

Integration with Game Loop

The Minimax algorithm seamlessly integrates into the game loop in `main.py`. During the AI player's turn (`game.turn == WHITE`), the algorithm calculates the optimal move and updates the game. Key points include:

- AI Move Calculation: The AI player's turn triggers the Minimax algorithm with a specified depth of 4 (`minimax(game.get_board(), 4, WHITE, game)`).

- Optimal Move Application: The optimal move is determined, and the game state is updated using the `ai_move` function.

The implementation of the Minimax algorithm in the Checkers game demonstrates its effectiveness in creating a challenging AI opponent. By evaluating potential future states and selecting moves strategically, the algorithm enhances the player experience, providing a compelling and engaging gameplay environment.

GUI

The Graphical User Interface (GUI) for this checkers AI project is implemented using the Pygame library. Pygame provides a powerful set of tools for handling graphics and user input, allowing for the creation of an interactive and visually appealing representation of the checkers game.

GUI Features:

Board Visualization:

The GUI displays the checkers game board with defined rows and columns. It uses colors to represent different elements, such as squares and pieces.

Piece Representation:

Each player's pieces are visually represented on the board, distinguishing between RED and WHITE pieces. The graphical representation enhances the user's understanding of the game state.

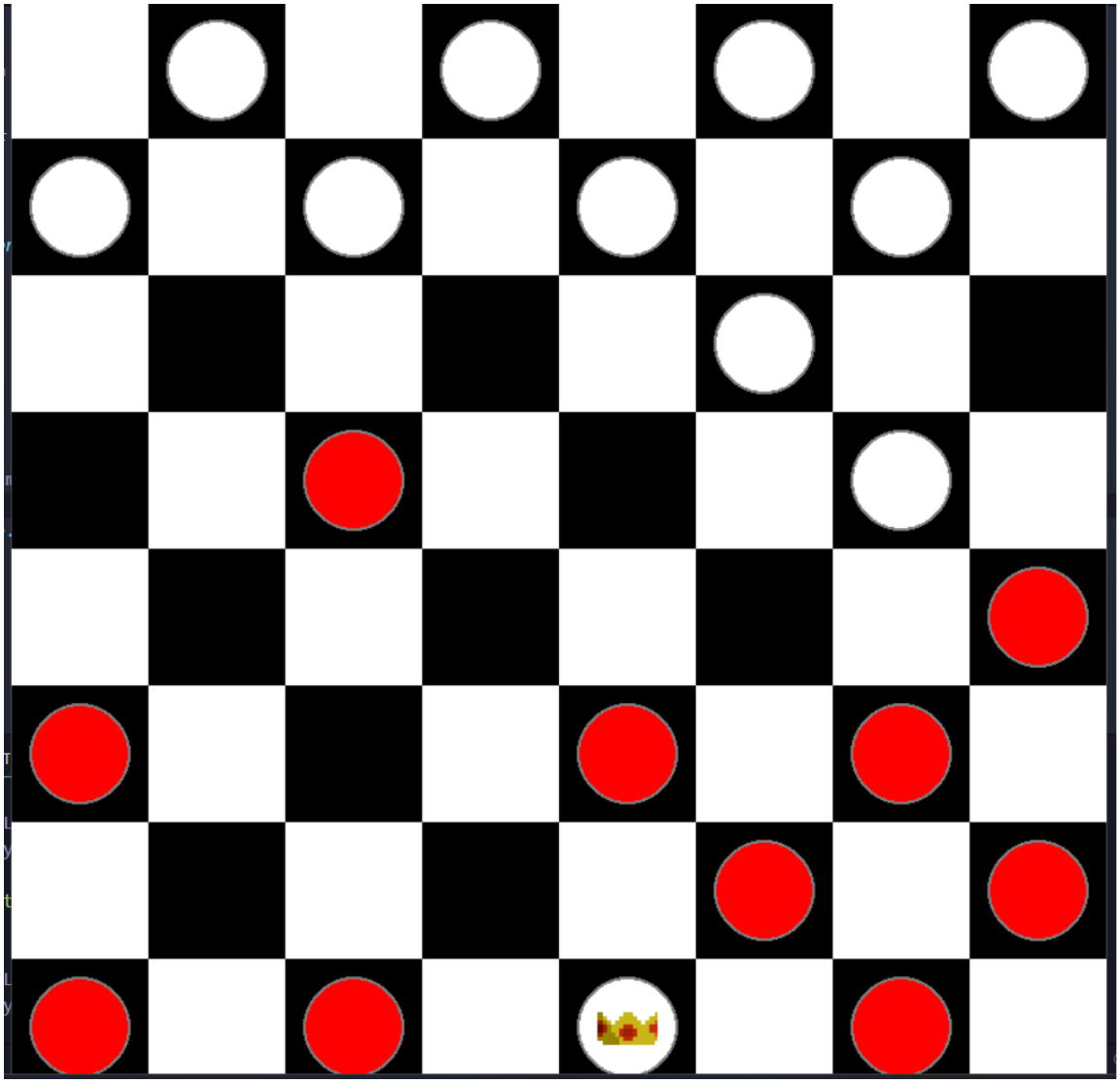
Interactive Elements:

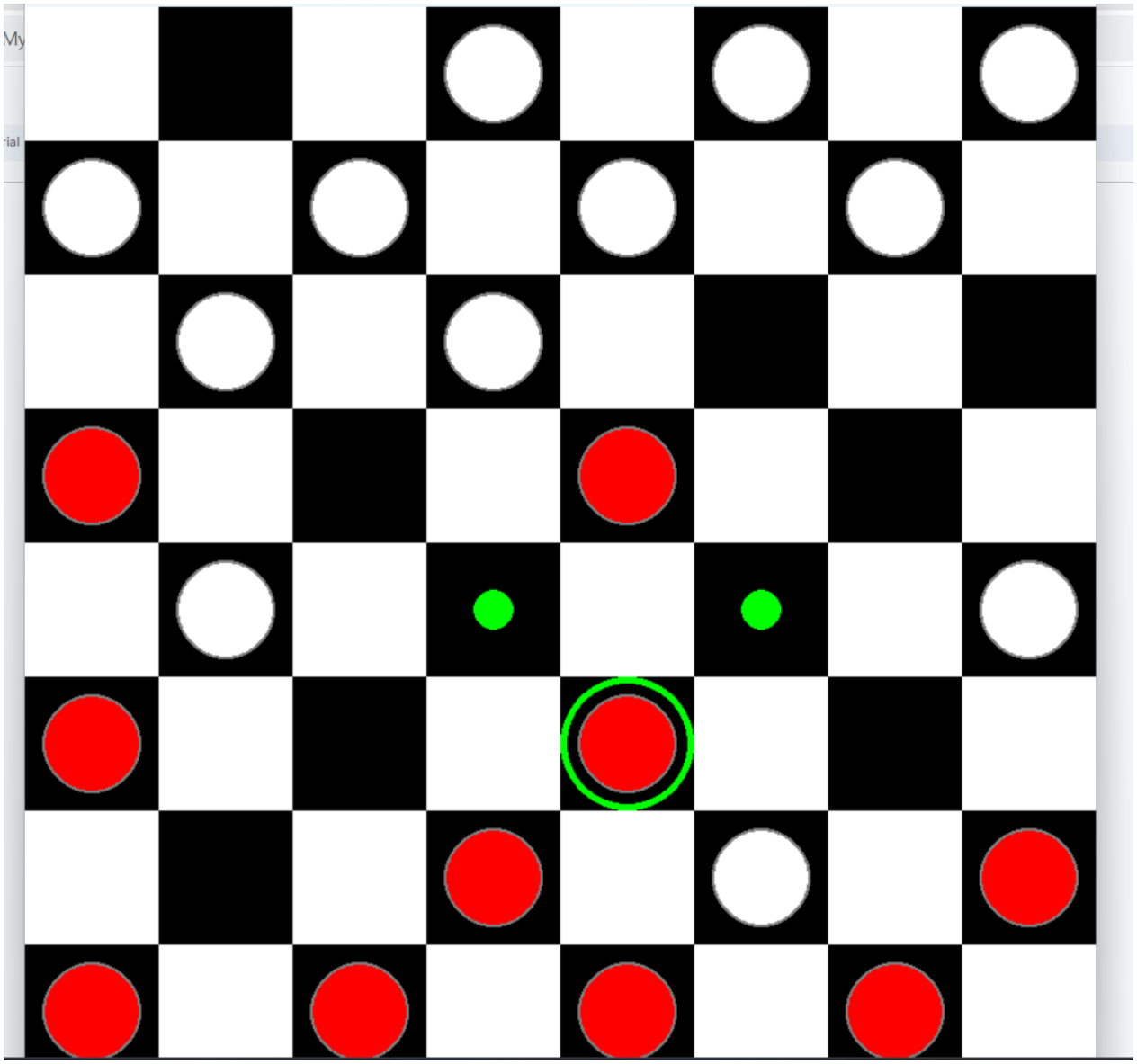
Users can interact with the GUI through mouse input to make moves and perform actions within the game. This interactive feature enhances the user experience and facilitates gameplay.

Pygame Integration:

Pygame is instrumental in the creation of the graphical interface, providing functions for drawing shapes, handling events, and updating the display. The GUI implementation enriches the checkers game, making it more accessible and engaging for users.

The integration of Pygame in this project not only ensures a visually appealing interface but also contributes to the overall user-friendliness of the checkers AI application.





LIBRARIES IMPORTED

- **Pygame:**

Utilized for graphical user interface development, providing functionalities for graphics handling and user input. Pygame is instrumental in creating an interactive representation of the checkers game.

- **Deep Copy Module (deepcopy):**

Imported the deepcopy function from the copy module for creating deep copies of objects. This is particularly useful when handling complex data structures and maintaining the integrity of the game state.

- **Constants Module (constants):**

Imported constants related to colors (BLACK, RED, WHITE, BLUE), square size (SQUARE_SIZE), rows (ROWS), columns (COLS), and additional constants for game logic (CROWN). Constants contribute to code readability and provide a centralized way to manage parameters.

- **Piece Module (Piece class):**

Imported the Piece class, which encapsulates the properties and behaviors of checkers pieces. This module enhances code modularity and encapsulates piece-related functionality.

- **Board Module (Board class):**

Imported the Board class from the checkers. board module, encapsulating the logic and functionality of the checkers game board. This modular approach enhances code organization and maintainability.

CONTRIBUTION OF INDIVIDUAL MEMBER

Tarushi Nimje- Board and Constants Developer:

- Spearheaded the creation of the game board and constants, laying the foundation for the game's structure.
- Implemented the checkered pattern, piece placement, and the definition of game constants crucial for gameplay consistency.

Mahima Patil- Game and Piece Developer:

- Led the development of the game logic, user interaction, and piece functionalities.
- Implemented the core mechanics of legal moves, captures, and king promotions, ensuring a dynamic and authentic gaming experience.

Rutuja Waghmode- Main and Algorithm Developer:

- Orchestrated the main game loop and user interface, bringing together the different components into a cohesive and playable game.
- Implemented the Minimax algorithm for the AI opponent, enhancing the game's strategic depth and providing a challenging opponent for players.

CONCLUSION

To conclude, this AI project centered on the implementation of the Minimax algorithm for playing checkers, emphasizing strategic decision-making in an adversarial game environment. Through extensive development and experimentation, we successfully created a checkers-playing agent capable of intelligent moves using the Minimax algorithm.

The absence of alpha-beta pruning in our approach showcased the fundamental strength of the Minimax algorithm in exploring the game tree and making optimal decisions. Despite the computational challenges associated with deeper search depths, the agent demonstrated competence in strategic play.

This project lays the groundwork for future research in checkers AI, emphasizing the simplicity and effectiveness of the Minimax algorithm.

In essence, our work contributes to the broader understanding of Minimax in checkers, providing a basis for future exploration and refinement of AI strategies in adversarial game scenarios.

REFERENCES

Artificial Intelligence | Mini-Max Algorithm - Javatpoint

Python Checkers AI Tutorial Part 1 - The Minimax Algorithm Explained - YouTube

https://www.247checkers.com/

https://www.geeksforgeeks.org/game-playing-in-artificial-intelligence/