AI CHECKERS GAME

Prepared for: Ms. Mehak Mazhar

BY:
MARYAM HASAN (22K-4701)
MUHAMMAD ARHAM ZULFIQAR (22K-4707)
MUHAMMAD HASSAN BAIG (22K-4696)

Introduction

This project involves a two-player Checkers game, with one player controlled by an AI and the other by a human. The AI makes intelligent decisions using the **Minimax algorithm**, a classic decision rule in AI game theory. The graphical interface is built using **Pygame**, offering interactive gameplay and a visually rich experience.

Objectives

- Develop a playable Checkers game with graphical interface.
- Integrate an AI agent capable of competing against a human.
- Utilize the **Minimax** algorithm for decision-making.
- Allow visualization of moves and game state updates in real time.

Tools and Technologies

- **Programming Language:** Python
- **Graphics Library:** Pygame
- **AI Algorithm: Minimax**with basic evaluation heuristics
- **IDE:** Visual Studio Code

System Design and Architecture

a. Module Overview

- **main.py**

Entry point of the program. Handles game loop, event handling, and calls AI moves.

- **game.py**

Manages the core gameplay mechanics—piece selection, turn switching, move execution, and AI interaction.

- **board.py**

Contains the board logic, including drawing the board, evaluating the board state, getting valid moves, and determining the winner.

- **Piece.py**

Defines how a piece is drawn and moved on the board. Also handles king-making.

- **algorithm.py**

Implements the **Minimax algorithm**, simulates potential moves, and selects the best move based on evaluation.

- **constants.py**

Stores constants like screen dimensions, colors, and piece sizes.

- **assets/**

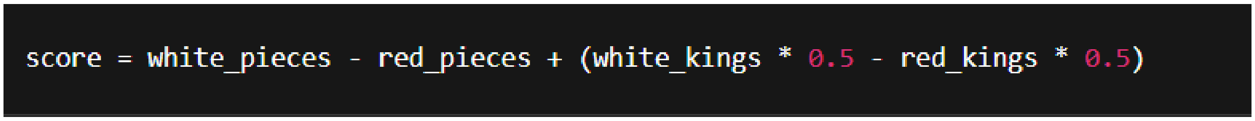
Contains the crown.png image used to visually identify king pieces.

- **readme.md**

Brief description of the project.

AI Implementation: Minimax Algorithm

The AI uses a basic **Minimax** algorithm with a fixed depth (set to 2) to simulate future game states and choose the most promising one. The evaluation function is simple:



```
score = white_pieces - red_pieces + (white_kings * 0.5 - red_kings * 0.5)
```

This function prioritizes material advantage and gives additional weight to king pieces.

Key Features:

- Recursive simulation of moves up to a specified depth.
- Maximizing and minimizing turns for the AI and human player.
- Move simulation and board cloning using deepcopy.

User Interface (UI)

The game interface is rendered using **Pygame**:

- 8x8 board with alternating-colored squares.
- Click-to-select and move pieces.
- Highlighted valid moves.
- Display of winner when the game ends.

Features

- Full Checkers gameplay rules (basic movement, captures, king promotion).
- AI opponent using *minimax*.
- Game reset and restart capability.
- Turn-based interaction with visual feedback.

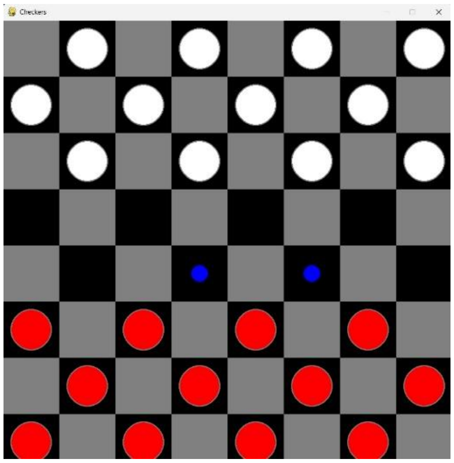
Limitations

- AI depth is fixed and limited (due to performance constraints).
- No undo or save/load game state.
- Basic evaluation function (can be improved for smarter play).

Future Work

- Add **alpha-beta pruning** to improve AI efficiency.
- Implement **difficulty levels** by varying the Minimax depth.
- Add multiplayer over network or online gameplay.
- Improve UI/UX (animations, sounds, menus).
- Implement enhanced evaluation functions (mobility, board control).

Screenshots

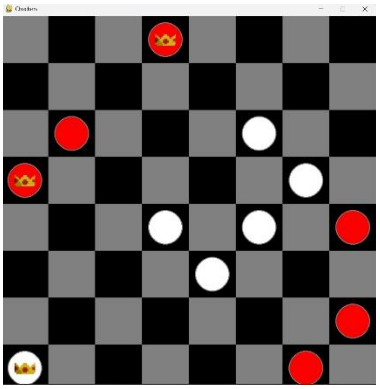


The initial gameplay setup where the red and white pieces are positioned on opposite sides of the board according to standard checkers rules.

1.<u>Red pieces</u> represent the human player's pieces.

2.<u>White pieces</u> represent the AI-controlled opponent.

3.<u>Two blue dots</u> in the center indicate **valid move options** for a selected red piece, highlighting possible destinations based on current rules and state.

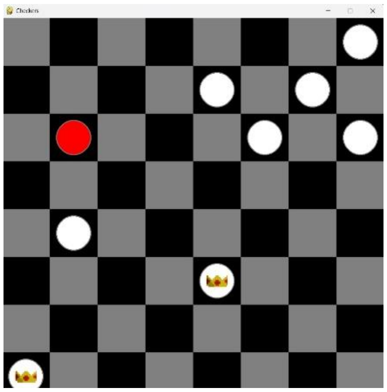


Human Wins!

Shows a game state where the **human player (Red)** has defeated the **AI (White).**

Several **red pieces have been promoted to kings**, indicated by the gold crown icons.

The AI is left with limited pieces and **no valid moves** on the board.



AI Wins!

The game state where the **AI player (White)** has achieved victory over the **human player (Red)**.

Most of the **red pieces have been captured**, with only one remaining on the board.

The **AI side has a strategic advantage** with multiple pieces in dominant positions, including **2 kinged pieces**.

Conclusion

This project demonstrates how classical AI algorithms like Minimax can be applied to real-world games. The game successfully integrates interactive gameplay with AI decision-making, offering both fun and a challenge to users. It serves as a solid foundation for future improvements in AI-based game development.

References

https://youtu.be/ipExjmyd6cc?si=mgA_e1k1l5YYeQ-c