Checkers Game with AI using Minimax Algorithm and Epsilon-Greedy Approach

Game Rules and Objective

Checkers, or draughts, is a strategic two-player board game played on an 8x8 grid. Each player starts with 12 pieces placed on the first three rows closest to them.Players alternate moves until one wins or no moves remain.

Pieces:

- Regular pieces move diagonally forward one step to an empty square. They can capture opponent pieces by jumping over them diagonally to an empty square. Multiple captures in one turn are allowed if conditions permit.
- Kings are created when a piece reaches the opponent's back row. Once promoted, kings can move and capture diagonally both forward and backward.

Winning:

Win by capturing all the opponent's pieces or blocking all their possible moves. The game ends in a draw if neither player can win.

Strategic Depth:

Checkers is easy to learn but requires strategic thinking to control the board, plan captures, and promote pieces to kings.

Gameplay Modes:

This project simulates a Checkers environment using the `Pygame` library, offering two gameplay modes:

- **Human vs Human**
- **Human vs AI**

In the **Human vs AI** mode, the difficulty level (Easy, Medium, or Hard) adjusts the depth of the Minimax algorithm and the randomness of moves using the epsilon-greedy approach.

Game Environment with Pygame

The `Pygame` library creates a visually interactive environment. It enables real-time rendering of the board, pieces, and animations, as well as handling user inputs. The game's mechanics follow standard Checkers rules, including diagonal moves, capturing pieces, and king promotions.

Minimax Algorithm

The Minimax algorithm is a decision-making strategy used in adversarial games. It assumes that both players will play optimally and aims to minimize the opponent's advantage while maximizing its own.

How Minimax Works

The algorithm explores all possible moves from the current game state by constructing a decision tree. It alternates between two players:

- **Maximizer:** Attempts to maximize the evaluation function, representing the AI's advantage.
- **Minimizer:** Attempts to minimize the Maximizer's advantage, simulating the opponent's strategy.

Each node in the tree represents a potential game state, and its value is determined by an evaluation function. The algorithm recursively evaluates these states up to a predefined depth, choosing the move that leads to the best possible outcome for the AI.

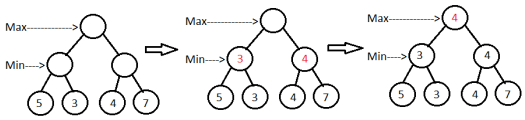

Figure 1: Visualisation of minimax algorithm

Evaluation Function

The evaluation function assigns a numeric score to each game state based on:

- The number of AI and opponent pieces.
- Kinged pieces, which have higher strategic value.

Player	Normal Piece	King
AI	+1	+3
Opponent	-1	-3

Table 1: Evaluation Scores for Pieces

Difficulty Levels

The depth of the Minimax algorithm is a key factor influencing difficulty:

- **Easy (Depth = 2):** Explores only the immediate moves, making decisions based on short-term gains.
- **Medium (Depth = 4):** Considers the opponent's response to the AI's moves, enabling more strategic play.
- **Hard (Depth = 6):** Evaluates multiple layers of possible moves, leading to highly strategic and competitive gameplay.

Epsilon-Greedy Approach

The epsilon-greedy approach introduces randomness into the AI's decisions, making gameplay less predictable:

- With a probability ϵ, the AI selects a random move (exploration).
- With a probability $1 - \epsilon$, the AI selects the best move determined by Minimax (exploitation).

Difficulty-Level Epsilon Values:

- **Easy:** High ϵ (0.2), resulting in frequent random moves.
- **Medium:** Moderate ϵ (0.1), balancing randomness and strategy.
- **Hard:** Low ϵ (0.05), focusing on calculated moves.

Game Modes

Human vs Human

Two players alternate turns on the same device. The game validates moves and enforces rules, such as capturing and king promotions.

Human vs AI

The player competes against an AI opponent. Before starting, the player selects the difficulty level. The AI adapts its strategy using the Minimax algorithm and the epsilon-greedy approach, offering a unique challenge at each level.

Conclusion

This project demonstrates the application of AI techniques in game development. By integrating the Minimax algorithm and the epsilon-greedy strategy, the AI provides dynamic and adaptive gameplay. The project highlights the power of AI in creating intelligent and engaging games, showcasing how game theory and randomness can be effectively combined to produce challenging AI opponents in a strategic setting.

Authors

Deepanjali Kumari - 22110069
Anura Mantri - 22110144
Bhanu Repalle - 22110221
Saloni Shinde - 22110242
Thumma Ushashree - 22110272

References

- Python Checkers AI
- Minimax Algorithm and Alpha-Beta Pruning
- Minimax Algorithm Visualisation