Project Report

Project Title: AI-Powered Strategic Checkers Game Using Minimax Algorithm
Submitted By: Taimoor Nazar
Course: AI
Instructor: Abdullah Yaqoob
Submission Date: 11/05/2025

1. Executive Summary

- **Project Overview:**

This project focuses on developing an AI-powered version of the traditional Checkers game. The core objective is to build a strategic AI opponent using the Minimax Algorithm to simulate intelligent gameplay. The AI analyzes current board states, predicts future outcomes, and makes optimal moves based on a heuristic evaluation function. Innovations include board state heuristics, AI decision-making, and optional Alpha-Beta pruning to optimize performance.

2. Introduction

- **Background:**
  Checkers is a classic two-player game played on an 8x8 grid, however, here it is done on a 10x10 grid, where players move diagonally and aim to capture all opponent pieces. This project was chosen to explore strategic decision-making in a deterministic game. To enhance gameplay, an AI component has been integrated using the Minimax Algorithm, enabling dynamic, intelligent opposition for solo players.
- **Objectives of the Project:**
  1. *Implement a playable version of Checkers with a responsive UI.*
  2. *Develop an AI using the Minimax Algorithm.*
  3. *Introduce a heuristic evaluation function for game states.*
  4. *Optimise performance with optional Alpha-Beta pruning.*
  5. *Test the AI against human players for effectiveness and difficulty.*

3. Game Description

- **Original Game Rules:**

1. *Played on an 8x8 board with alternating dark and light squares.*

2. *Each player begins with 12 pieces on the dark squares of the first three rows.*
3. *Pieces move diagonally forward and capture by jumping.*
4. *Kings (promoted pieces) can move backward as well.*
5. *The game ends when a player loses all pieces or can no longer move.*

- **Innovations and Modifications:**

□ *AI opponent using Minimax to determine the best move.*
□ *Board evaluation considers piece count, kings, and positioning.*
□ *Move simulation enables lookahead strategy.*
□ *Forced capture rule enforced programmatically.*
□ *Optional Alpha-Beta pruning for performance enhancement.*

4. AI Approach and Methodology

- **AI Techniques Used:**

  □ ***Minimax Algorithm:*** *Evaluates all possible game states recursively to choose the optimal move.*
  □ ***Alpha-Beta Pruning (Optional):*** *Reduces the number of nodes evaluated in Minimax by pruning unneeded branches.*

- **Algorithm and Heuristic Design:**

  1. ***Piece Value:*** *Regular = 1 point, King = 3 points.*
  2. ***Positional Advantage:*** *Points for center control and safe positions.*
  3. ***Capture Opportunities:*** *Moves leading to captures are prioritized.*
  4. *The evaluation function combines these metrics to assign a score to each board state.*

- **AI Performance Evaluation:**

  □ *AI tested against human players of varying skill.*
  □ *Metrics: win rate, average decision time, and number of moves considered.*

5. Game Mechanics and Rules

- **Modified Game Rules:**

  □ *Single-player mode where AI plays as one of the players.*
  □ *Forced capture if a jump is available.*
  □ *Promotion to King when a piece reaches the opponent's baseline.*

□ *Kings can move both forward and backward.*
□ *On a 10x10 grid*

- **Turn-based Mechanics:**

□ *Turns alternate between player and AI.*
□ *After each player move, the AI evaluates and makes its move automatically*

- **Winning Conditions:**
  A player wins if the opponent has no legal moves or all pieces are captured.

6. Implementation and Development

- **Development Process:**

  □ *Designed game logic and UI using Python.*
  □ *Implemented Minimax algorithm with recursive move evaluation.*
  □ *Integrated board evaluation heuristics.*
  □ *Conducted iterative testing and debugging.*

- **Programming Languages and Tools:**

  □ ***Programming Language:*** *Python*
  □ ***Libraries:*** *Pygame (for game interface), NumPy (for matrix handling, optional)*
  □ ***Tools:*** *GitHub (version control)*

- **Challenges Encountered:**

  □ *Ensuring efficient move generation and backtracking for Minimax.*
  □ *Preventing performance lags during deeper searches.*
  □ *Balancing AI strength and response time.*

7. Team Contributions

- **Team Members and Responsibilities:**

  [Taimoor]: Implemented Minimax and Alpha-Beta pruning.

  [Rafey]: Developed game UI using Pygame.

  [Urwa]: Worked on heuristic function design and board evaluation logic.

8. Results and Discussion

- **AI Performance:**
    - □ *The AI demonstrated consistent strategic play and outperformed beginner human players in over 80% of test games.*
    - □ *Decision time averaged 1.5–3 seconds per move with Alpha-Beta pruning.*
    - □ *The heuristic evaluation effectively guided decision-making, especially in endgame scenarios.*

9. References

- *Russell, S., & Norvig, P. (2020). Artificial Intelligence: A Modern Approach.*
- *Checkers game rules: Wikipedia*
- *Minimax and Alpha-Beta pruning tutorials: GeeksforGeeks, Stack Overflow*