Project Report

Project Title: *Checkers AI*
Submitted By: Sarim Shah(22K4299), Moiz Ul Haq(22K4587), Muhammad Rouhan (22K4577)
Course: AI
Instructor: Sir Shafique Rehman
Submission Date: 5/3/2025

1. Executive Summary

● Project Overview:

This project aims to develop an intelligent agent which can play Checkers using a combination of Reinforcement Learning (specifically Deep Q-Learning) and traditional game tree search techniques such as Alpha-Beta Pruning. The game includes a user-friendly interface built with Pygame, allowing users to play against the AI. The Q-AI agent is trained by self-play and continuously improves its strategy over time, making it adaptive and increasingly competitive.

2. Introduction

● Background:

Checkers is a well-known two-player strategy board game that involves diagonal movement and capturing opponent pieces by jumping over them. It was chosen due to its balance of complexity and learnability, making it ideal for reinforcement learning experiments. Unlike chess, the rules are simpler but still provide rich strategic depth. Our project modernizes Checkers by incorporating a learning AI that adapts its strategy based on experience, offering a smarter and more challenging opponent.

● Objectives of the Project:

- *Develop a Reinforcement Learning-based AI agent to play Checkers.*
- *Optimize AI decision-making with Alpha-Beta Pruning.*
- *Create an interactive and visually intuitive user interface.*
- *Evaluate AI performance based on win rate and move efficiency.*

3. Game Description

● *Original Game Rules:*

Checkers is played on an 8x8 board with 12 pieces per player. Players take turns moving diagonally forward. Captures are made by jumping over opposing pieces. Kings (pieces that reach the opposite end) can move backward and forward. The game ends when a player has no legal moves left. But this checkers has 8x12 board and has 18 pieces per player as this was the requirement of the project.

● *Innovations and Modifications:*

- *Integration of an AI opponent using Q-learning.*
- *Option to play against a heuristic-based or learning-based AI.*
- *Real-time performance tracking (win/loss ratio).*
- *Enhanced UI built with Pygame for better interaction and feedback.*

4. AI Approach and Methodology

● *AI Techniques Used:*

- ***Reinforcement Learning**: Deep Q-learning was used to enable the agent to learn optimal strategies through exploration and reward feedback.*
- ***Alpha-Beta Pruning**: Traditional game tree search was optimized to evaluate board positions up to a specified depth.*

● *Algorithm and Heuristic Design:*

The Q-learning model uses a neural network to approximate the Q-values for game states. State representation includes the board configuration. For Alpha-Beta, a simple heuristic evaluates states based on advantages (piece count and king count).

● *AI Performance Evaluation:*

The agent's performance was evaluated based on:

- *Win rate over 100+ test games.*
- *Average decision-making time per move.*
- *Ability to adapt to human strategies over time.*

5. Game Mechanics and Rules

Modified Game Rules:

- *Double and multiple jumps are supported.*
- *Forced capture rule enforced.*
- *Bigger board 8x12 and more pieces per player i.e 18*

Turn-based Mechanics:

Players alternate turns. The game checks for valid moves . The AI is triggered to respond after each human move.

Winning Conditions:

A player wins when the opponent has no remaining pieces or legal moves. Draws are possible when neither side can force a win.

6. Implementation and Development

⬤ *Development Process:*

The game was developed in Python using Pygame for the graphical interface. TensorFlow was used to implement and train the Q-learning model. The board logic, game state management, and AI models were developed in modular components.

⬤ *Programming Languages and Tools:*

- ***Language**: Python*
- ***Libraries**: Pygame, NumPy, TensorFlow*
- ***Tools**: VS Code (development environment)*

⬤ *Challenges Encountered:*

- *Designing a suitable state representation for Q-learning.*
- *Handling edge cases in multi-jump rules.*
- *Balancing training time and model performance.*
- *Managing game state transitions efficiently in the AI loop.*

7. Team Contributions

⬤ *Sarim Shah (22K4299):*

- *Developed and trained the Q-learning AI using TensorFlow.*
- *Implemented state representation and Q-value approximation.*
- *Assisted in AI performance evaluation and tuning.*

⬤ *Moiz Ul Haq (22K4587):*

- *Designed and implemented game logic and rule enforcement.*

- *Created and refined the Pygame-based user interface.*
- *Integrated AI decisions flow into the UI game loop.*

● *Muhammad Rouhan (22K4577):*

- *Developed Alpha-Beta pruning logic for rule-based AI.*
- *Handled model saving/loading and performance tracking.*
- *Led performance evaluation, documentation, and testing.*

8. Results and Discussion

● *AI Performance:*

- *The Deep Q-learning agent achieved a **win rate of 20%** against human players after sufficient training. (not trained well because of resource issue)*
- *Average QAI move decision time: **5-10 seconds**, depending on model complexity and board state. And Minimax Agent move decision time is within a second.*
- *The AI demonstrated adaptive behavior and improved gameplay quality over time.*

9. References

- *Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction.*
- *Pygame documentation: https://www.pygame.org/docs/*

- *TensorFlow documentation: https://www.tensorflow.org/api_docs*