**🧠 Project Report**

**📌 Project Title: Checkers AI**

**Submitted by:** Waleed Haroon (22K-4596), Ahmed Iltaf  
 **Course:** Artificial Intelligence

Instructor:Alina arshad , Ravia ejaz

Submission Date: 15th may 2025

**1. Executive Summary**

***🔍 Project Overview***

This project focuses on developing an intelligent agent capable of playing Checkers using a combination of **Deep Q-Learning** and **Alpha-Beta Pruning**. The agent is designed to learn and improve through self-play, making its gameplay adaptive over time. The game includes an interactive **Pygame-based GUI** that allows users to play against the AI, offering an engaging and challenging experience.

**2. Introduction**

***📚 Background***

Checkers is a classic two-player strategy board game involving diagonal movements and capturing opponent pieces by jumping. It offers the perfect blend of simplicity and depth, making it suitable for experimenting with reinforcement learning techniques. Unlike chess, Checkers has more manageable rules but still requires thoughtful strategy, making it an ideal testbed for AI development.

***🎯 Project Objectives***

* Develop a Reinforcement Learning-based AI agent for Checkers
* Integrate Alpha-Beta Pruning to optimize decision-making
* Create an intuitive graphical interface using Pygame
* Evaluate AI performance through win rates and decision efficiency

**3. Game Description**

***♟️ Game Rules (Modified)***

* Board Size: 8x12 (instead of standard 8x8)
* Pieces: 18 per player
* Movement: Diagonal moves only
* Capturing: Jump over opponent pieces; multi-jumps allowed
* Kings: Pieces reaching the opposite end become kings and can move both directions
* Endgame: Game ends when a player has no legal moves or pieces

***⚙️ Innovations & Features***

* AI opponent powered by Q-learning
* Option to play against heuristic-based (Alpha-Beta) or learning-based (Q-learning) AI
* Real-time performance tracking (win/loss ratio)
* Enhanced, user-friendly Pygame interface

**4. AI Approach and Methodology**

***🤖 AI Techniques***

* **Deep Q-Learning:** Enables the agent to learn from its experience by using a neural network to approximate Q-values.
* **Alpha-Beta Pruning:** An optimization over Minimax to reduce the number of nodes evaluated in the game tree.

***🧠 Algorithm & Heuristic Design***

* **State Representation:** Encodes the board configuration as input to the neural network.
* **Q-Network:** Learns optimal actions based on rewards.
* **Alpha-Beta Heuristic:** Evaluates game states based on piece count, king count, and board control.

***📊 Performance Evaluation***

* Win rate over 100+ test games
* Average move decision time
* Ability to adapt against human opponents

**5. Game Mechanics and Rules**

***🔄 Turn-Based Flow***

* Players alternate turns
* After a human move, the AI computes and responds automatically
* All valid moves, including forced captures, are enforced

***🎯 Winning Conditions***

* Opponent has no legal moves or pieces left
* Draws occur when no progress can be made by either player

**6. Implementation and Development**

***🛠️ Development Overview***

The game was implemented in **Python** with **Pygame** for the interface and **TensorFlow** for training the Q-learning model. A modular design separates logic, UI, and AI layers, ensuring code maintainability and clarity.

***🧰 Tools & Technologies***

* **Language:** Python
* **Libraries:** Pygame, NumPy, TensorFlow
* **IDE:** Visual Studio Code

***🚧 Challenges Faced***

* Designing effective state representations
* Managing multi-jump edge cases
* Balancing training time with available computational resources
* Efficient integration of AI decisions into gameplay

**7. Team Contributions**

* **Waleed Haroon (22K-4596):**
* Developed the Deep Q-learning agent using TensorFlow
* Designed the state representation and reward mechanism
* Evaluated AI performance and handled fine-tuning
* **Ahmed Iltaf:**
* Designed the game logic and enforced rules
* Built and refined the Pygame graphical interface
* Integrated AI decision-making into the game loop
* Implemented Alpha-Beta pruning for heuristic-based AI

**8. Results and Discussion**

***📈 AI Performance***

* **Q-Learning Agent Win Rate:** ~20% against human players (limited training due to resource constraints)
* **Decision Time:**
* Q-learning agent: 5–10 seconds per move
* Alpha-Beta agent: <1 second per move
* **Adaptability:** AI demonstrated progressive improvement and adaptability through training

**9. References**

* Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*
* Pygame Documentation: <https://www.pygame.org/docs/>
* TensorFlow API: <https://www.tensorflow.org/api_docs>