AI Checkers Game - Project Report

1. Introduction

1.1 Why This Project Was Developed

The motivation behind developing this AI-powered Checkers game was to explore the practical application of reinforcement learning in-game environments. Traditional games like Checkers offer a well-defined rule set and measurable outcomes, making them ideal for AI experimentation. Our goal was to go beyond a simple implementation and build an intelligent, self-learning agent that can play Checkers effectively while supporting human interaction and multiple game modes. We also aimed to present a complete software product that combines machine learning, game development, and user interface design.

2. What the Project Does

The AI Checkers game is a Python-based application that:

- Allows users to play Checkers in three modes:
  - Human vs AI
  - Human vs Human
  - AI vs AI
- Offers multiple difficulty levels: Easy, Medium, and Hard
- Includes a self-play training mode where two AI agents learn by playing against each other
- Tracks live statistics such as games played, win/loss/tie count, and average number of moves
- Displays a GUI using Pygame with features such as turn indicators, valid move highlights, and endgame messages

3. How It Works

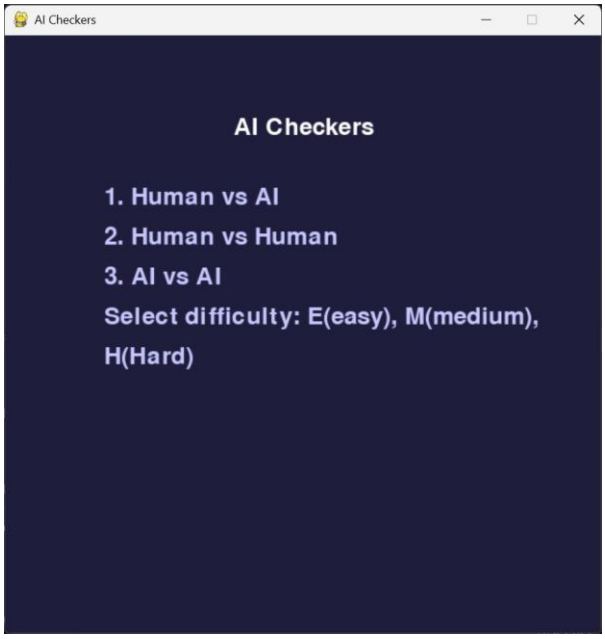
3.1 Game Engine

The game engine is built using Python and Pygame. The Board and Piece classes define the rules and state of the game. Legal moves, captures, king promotions, and win conditions are handled entirely in Python logic.

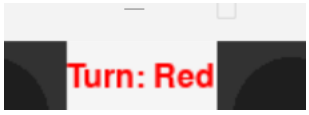
3.2 User Interface

The UI is rendered using Pygame. Players interact via mouse clicks, and the interface includes:

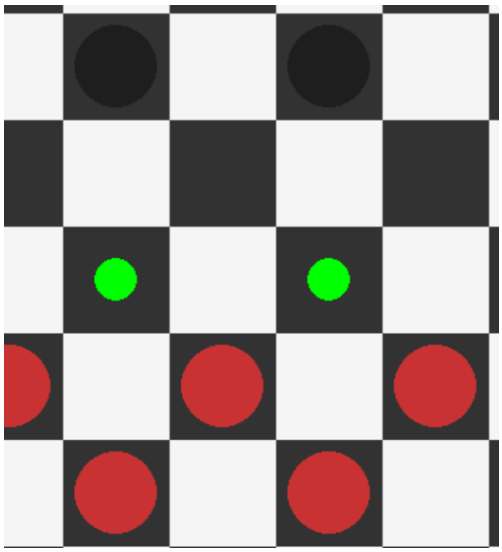
- A menu screen for selecting mode and difficulty



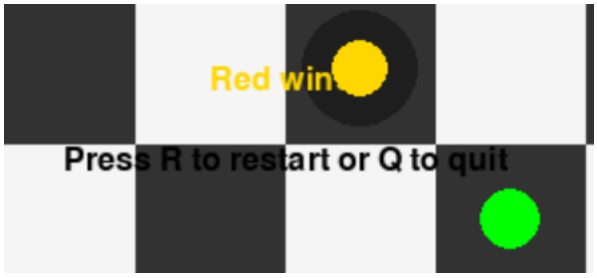
- Real-time turn display

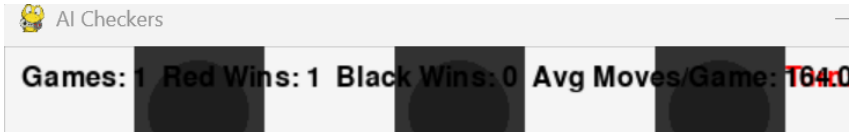


- Highlighted valid moves



- Endgame summary with restart and quit options





3.3 AI Agent (QLearningAgent)

We used the Q-learning algorithm for our AI agents:

- States are encoded from the board into a string format
  - Actions are defined as move transitions (e.g., 2,3->3,4)
- Rewards are given for:
  - +10 per captured piece
  - +100 for winning
  - -100 for losing
- Q-values are updated using the Bellman equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max Q(s', a') - Q(s, a)]$$

- An ε-greedy policy is used to balance exploration vs exploitation
  - Epsilon decays over time to allow the AI to become more deterministic

3.4 Self-Play Training Mode

In this mode, two AI agents (Red and Black) play thousands of episodes against each other to improve their strategies. The Q-values are saved into `.pkl` files and loaded during gameplay to enable intelligent decision-making. Training logs show that exploration drops as the agents converge.

```
[Running] python -u "c:\Users\hgcfm\OneDrive\Desktop\AI Checkers Project\train.py"
pygame 2.6.1 (SDL 2.28.4, Python 3.13.0)
Hello from the pygame community. https://www.pygame.org/contribute.html
Episode 500/4500 - eps red: 0.281, black: 0.290
Episode 1000/4500 - eps red: 0.077, black: 0.087
Episode 1500/4500 - eps red: 0.050, black: 0.050
Episode 2000/4500 - eps red: 0.050, black: 0.050
Episode 2500/4500 - eps red: 0.050, black: 0.050
Episode 3000/4500 - eps red: 0.050, black: 0.050
Episode 3500/4500 - eps red: 0.050, black: 0.050
Episode 4000/4500 - eps red: 0.050, black: 0.050
Episode 4500/4500 - eps red: 0.050, black: 0.050
Training complete. Q-tables saved.
```

4. Implementation Details

4.1 Technologies Used

- Python 3.11
- Pygame for rendering GUI
- NumPy for Q-value data handling

- Pickle for saving/loading trained Q-tables

4.2 File Structure

```
├── main.py
├── ai.py
├── board.py
├── piece.py
├── train.py
├── q_black.pkl
├── q_red.pkl
├── README.md
├── Project Proposal.docx
├── Project Report.pdf
├── Demo.mp4
```

5. Uniqueness and Innovation

Compared to other Checkers projects, our game stands out because:

- It combines reinforcement learning with real-time human interaction instead of using MiniMax and Alpha Beta Pruning
- It supports multiple difficulty levels using a single learned policy
- It includes an interactive menu system and polished GUI
- The AI agents continuously improve via training, unlike static rule-based bots
- A live statistics dashboard provides real-time feedback on AI and player performance

6. Contributions

Rida Qazi (22k-4409)

- Implemented the Q-learning algorithm and training logic
- Handled reward shaping and Q-value updates
- Tuned hyperparameters for convergence

Mashal Jawed (22k-4552)

- Designed the GUI layout and interface
- Integrated user inputs and display elements (turns, move highlights)
- Worked on menu design and navigation flow

Abdul Wasey (22k-4172)

- Developed the core game mechanics (move validation, board updates)
- Structured the project architecture and modular file system
- Added statistics tracking and AI vs AI automation

7. Conclusion

The AI Checkers game demonstrates how reinforcement learning can be integrated into a traditional game environment to create a dynamic and intelligent opponent. The modular design, extensive feature set, and user-friendly interface make this a comprehensive project that goes beyond basic gameplay.

We successfully implemented multiple game modes, trained an AI through self-play, and provided an engaging UI that brings it all together. The project showcases our ability to combine machine learning concepts with game development, creating a well-rounded and innovative solution.