**Project Report**

**Project Title:** Chinese Checkers

**Submitted By:** Yusra Asim (22k-4378) and Lamia Asif (22k-4622)

**Course:** AI

**Instructor:** Abdullah Yaqoob

**Submission Date:** 5/11/2025

**1. Executive Summary**

● **Project Overview:**

This project modifies traditional Chinese Checkers to support both 2-player and 4-player modes while introducing a novel blocking mechanic. The game incorporates a Minimax AI with Alpha-Beta pruning to enable competitive gameplay against human opponents. Key innovations include:

* Flexible player configuration (2 or 4 players)
* Blocking pieces that temporarily obstruct opponents
* Optimized AI decision-making with custom heuristics

**2. Introduction**

● **Background:**

Chinese Checkers is a classic strategy game where players race to move their pieces across a star-shaped board. This project reimagines the game with:

1. **Player Flexibility**: Supports both 2-player (head-to-head) and 4-player (cross-shaped) configurations.
2. **Strategic Blocking**: Introduces immobile blocking pieces that prevent opponents from jumping over them for one turn.

● **Objectives of the Project:**

* Develop a stable 2/4-player game engine
* Implement blocking mechanics without disrupting core gameplay
* Create an AI that adapts to both game modes
* Evaluate AI performance against human players

**3. Game Description**

● **Original Game Rules:**

Chinese Checkers is a strategic, turn-based board game played on a hexagram (six-pointed star) board with up to six players. Each player controls a set of pieces of one color, starting from one of the triangular “home” zones on the board. The goal is to move all of your pieces across the board into the opposing triangle, following specific movement rules. The board consists of 121 spaces arranged in a grid, allowing pieces to move in six possible directions.

● **Innovations and Modifications:**

Introducing 2 players or 4 players to play in turns on a diamond shaped board. An additional piece that can not move itself and blocks opponent's moves until next turn (opponent's piece can not jump over blocking piece)

This will add a new strategic layer and allow players to choose between moving their own piece or blocking opponent

**4. AI Approach and Methodology**

● **AI Techniques Used:**

We used the Minimax algorithm with Alpha-Beta pruning to allow the AI play as a player. It evaluates multiple possible moves and selects the optimal one in a competitive 2-4-player scenario.

● **Algorithm and Heuristic Design:**

WeightedHeuristic([

(SumOfPegsInCornerHeuristic(), 0.1),

(AverageManhattanToCornerHeuristic(), 0.3),

(AverageEuclideanToCornerHeuristic(), 0.4),

(MaxManhattanToCornerHeuristic(), 0.2)

])

● **AI Performance Evaluation:**

(Discuss how the AI's performance was evaluated, such as win rate, decision time, and accuracy.)

**5. Game Mechanics and Rules**

### Key Rules

1. **Blocking Pieces**
   * Placed during setup in each home triangle
   * Cannot move but blocks jumps until owner's next turn
   * Shown with special highlighting (red border)
2. **Movement Validation**

def validate\_jump():

if blocked\_path\_exists(src, dest):

return False

# Standard jump validation...

**Turn-based Mechanics:**

1. Human/AI chooses: Move piece **or** activate blocking
2. Automated turn rotation (1→2→3→4→1 in 4-player mode)

● **Winning Conditions:**

Win detected when all pieces reach opposite triangle

**6. Implementation and Development**

● **Development Process:**

The game was developed using Python 3.9 as the core engine, with Pygame handling the graphical interface. The AI framework is built on a custom Minimax algorithm with Alpha-Beta pruning, while analytics are managed using JSON for data storage and Matplotlib for performance visualization.

● **Programming Languages and Tools:**

* Board.py: Manages board initialization for both 2-player and 4-player modes, including terminal state checks to determine wins.
* GameController.py: Orchestrates turn flow, blocking mechanics, and player transitions while maintaining game state consistency.
* MinimaxAIPlayer.py: Implements the AI’s decision-making using Minimax with Alpha-Beta pruning, optimizing move selection under time constraints.

● **Challenges Encountered:**

#### 1. Player Turn Management in Multiplayer Mode

* Problem: The game initially crashed when switching between 2-player and 4-player modes due to incorrect player indexing.
* Solution: Implemented a modulo-based system (player\_index = (state.player - 1) % len(self.players)) to ensure valid player access.

#### 2. Blocking Mechanics Implementation

* Problem: Blocking pieces sometimes failed to prevent jumps due to improper path validation.
* Solution: Enhanced Step.validate\_jump() to check for blocking pieces in the movement path before approving jumps.

**7. Team Contributions**

● **Team Members and Responsibilities:**

○ **Yusra Asim:** Initial phase of game, board, player setup and graphics and original rules.

○ **Lamia Asif:** Handled game rule modifications and heuristics(minimax).

**8. Results and Discussion**

● **AI Performance:**

The AI won 100% of the matches in both 2 and 4 player formats.

**9. References**

* Minimax & Alpha-Beta Pruning
  + [Stanford CS221: Minimax Guide](https://stanford.edu/~shervine/teaching/cs-221/cheatsheet-adversarial-search)
  + [GeeksforGeeks: Alpha-Beta Pruning](https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/)
* Optimization
  + [Optimizing a Game of Chinese Checkers](https://vc.bridgew.edu/cgi/viewcontent.cgi?referer=&httpsredir=1&article=1121&context=honors_proj)
* Chinese Checkers Variations
  + [Original Rules (BoardGameGeek)](https://boardgamegeek.com/boardgame/2393/chinese-checkers)
  + [Hexagonal Grid Math (Red Blob Games)](https://www.redblobgames.com/grids/hexagons/)
* Blocking Mechanic Inspiration
  + [Halma Strategy (Similar Game)](https://www.ultraboardgames.com/halma/game-rules.php)