Final Project Report: AI-Powered Halma Game

Course: Artificial Intelligence

Instructor: Ms. Mehak Mazhar

Submission Date: 10/05/2025

Project Title

AI-Powered Halma with Strategic AI

Submitted By

Muhammad Minhal Mahmud (K21-3618)
Group Members:

- Huzaifa Farraz (K21-3602)
- Muhammad Abbas (K21-3592)
- Arham Asher (K21-3578)

1. Introduction

This project focuses on the development of an AI-based Halma game implemented in Python. The game integrates a strategic AI opponent that uses the Minimax algorithm with Alpha-Beta Pruning, enabling challenging gameplay against a human player. The implementation provides a visual, interactive interface using the Pygame library and applies multiple heuristics to evaluate the board state effectively.

2. Objective

The primary goal of this project is to build an intelligent Halma game where:

- An AI agent competes with a human player using the Minimax decision-making algorithm.
- Heuristics guide the AI in choosing optimal moves.
- A graphical user interface (GUI) enhances user interaction.
- The AI behavior can be tuned for difficulty through depth adjustment.

3. Game Overview

Original Game Background

Halma is a two-player strategic board game where each player aims to move all their pieces to the opponent's starting zone. Movement can be one step in any direction or by jumping over other pieces. The game emphasizes foresight, blocking strategies, and piece mobility.

Game Innovations

The following AI enhancements were integrated:

- **Minimax Algorithm with Alpha-Beta Pruning** for efficient decision-making.
- **Heuristic Evaluation** based on:
  - Distance to the target zone.
  - Board control and blocking.
  - Penalization of idle pieces.
- **Visual feedback** for both human and AI moves.
- **Difficulty adjustment** through Minimax depth configuration.

4. Technical Implementation

Language & Libraries

- **Language:** Python
- **Libraries Used:**
  - **Pygame** – for game visualization and event handling.
  - **Copy** – to create deep copies of board states during Minimax recursion.
  - **Math** – for distance calculations in heuristics.

AI Methodology

Minimax Algorithm

- Explores all possible future game states to determine the optimal move.
- Alternates between minimizing the human player’s score and maximizing the AI’s score.

Alpha-Beta Pruning

- Optimizes Minimax by ignoring branches that cannot influence the final decision.
- Reduces computational time significantly at deeper levels.

Heuristic Function

The heuristic function evaluates the game state based on:

- **Target Proximity**: Pieces closer to the opponent’s base score higher.
- **Zone Control**: AI prefers moves that block or occupy the opponent’s region.
- **Piece Activity**: Idle pieces are penalized to encourage consistent progression.

5. Game Mechanics

Rules Implemented

- Players take turns: Human clicks to move; AI responds with a calculated move.
- AI calculates the best move using Minimax with a specified depth (e.g., depth = 2).
- The player who successfully moves all pieces to the opponent's starting zone wins.
- AI wins if it blocks all possible human moves.

Winning Condition

Implemented using a method that checks if all a player’s pieces have reached the opposite corner or if the opponent has no valid moves left.

Turn Handling

In `main.py`, a loop tracks the game's turn:

- AI executes its move if it's the AI's turn.
- Player input is processed through mouse click events.
- The winner is checked after every move using `game.winner()` and a `game_over` flag prevents redundant execution after a winner is declared.

6. Key Code Explanation

`main.py` Highlights:

- Sets up a Pygame window.
- Handles user input via mouse clicks.
- Calls the `minimax` function to determine the AI’s move.
- Uses `game.winner()` and a `game_over` flag to handle game termination cleanly.

Sample Logic (from `main.py`):

```
if game.turn == WHITE and not game_over:
    value, new_board = minimax(game.get_board(), 2, float('-inf'),
float('inf'), WHITE, game)
    game.ai_move(new_board)
```

This block ensures the AI only makes a move if it's its turn and the game isn't over.

7. Testing and Evaluation

The game was tested with different depths for Minimax:

- **Depth 1**: Fast but less strategic.
- **Depth 2**: Balanced performance and strategy.
- **Depth 3+**: Better strategy but slower response time.

Observations:

- AI successfully blocks opponent strategies when depth ≥ 2.
- Heuristic tweaks significantly influence AI aggressiveness and defense.
- Alpha-Beta pruning provides noticeable performance gains.

8. Challenges Faced

- **AI Delay & Complexity**: Balancing depth for speed vs. intelligence was challenging.
- **Game Loop Bugs**: Repeated win prints required flags to fix infinite print loops.
- **Heuristic Design**: Multiple iterations were needed to make the AI play logically.

9. Future Work

- **Multiplayer Mode**: Add support for two human players.
- **Enhanced Heuristics**: Incorporate learning-based methods (e.g., Reinforcement Learning).
- **Move Animation**: Smooth transitions for better user experience.
- **Dynamic Difficulty**: Adjust AI depth dynamically during gameplay.

10. Conclusion

This project successfully demonstrates how classical AI techniques like Minimax and Alpha-Beta Pruning can be applied to a strategic board game like Halma. Through heuristic-based evaluation and an interactive GUI, the game provides an engaging experience, offering both challenge and insight into AI-based decision-making in turn-based games.

11. References

- Russell, S., & Norvig, P. *Artificial Intelligence: A Modern Approach*
- Halma Rules & Strategy Guides (Online sources)
- Pygame Documentation: https://www.pygame.org/docs/
- Research on Minimax and Alpha-Beta Pruning (IEEE and academic journals)