Minimax agent for HexThello game

Authors: Thomas Lagkalis
@Technical University of Crete

February 8, 2025

Abstract

This report presents the development of a Minimax-based AI agent for HexThello, a strategy board game. The agent utilizes the Minimax algorithm with Alpha-Beta pruning to efficiently evaluate possible moves and determine optimal strategies. The evaluation function incorporates key heuristics, including piece count, mobility, border control, and frontier stability, to evaluate board states. Additionally, move ordering techniques are implemented to enhance pruning efficiency. The agent was tested against different difficulty levels and human players to analyze its decision-making and adaptability. Experimental results highlight the effectiveness of the implemented heuristics, while challenges such as search depth limitations are discussed.

1 Introduction

HexOthello is a board game played by two players. Each player has pieces of a color (e.g. black and white) and the objective is to end up with more pieces than the opponent. Both players take turns placing discs on the hexagonal board to convert their opponent's previously placed discs. Dark plays first. At each turn the player must place a disc on an empty tile such that it is neighbor of at least one of it's opponent's discs. After placing it's disc all newly bound discs belonging to the opponent will converted to it's color.

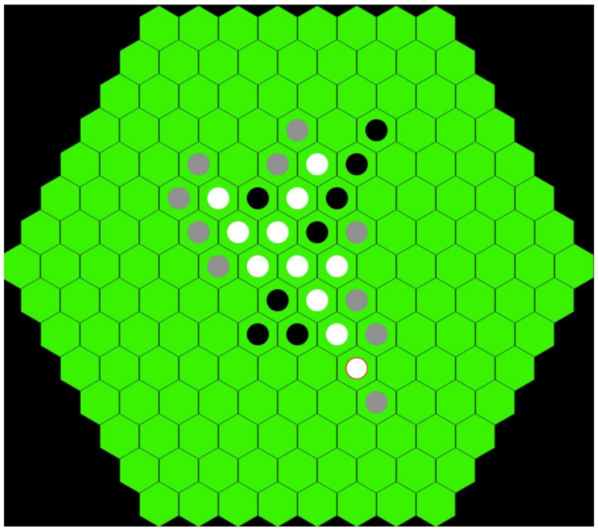

Figure 1: HexOthello board

First we introduce the evaluation function with all it's heuristics used for the minimax algorithm. Then we dive deepper in the minimax algorithm explaining the alpha-beta pruning, move ordering and Iterative Deepening Search (IDS) heuristics.

2 Evaluation function

The evaluation [2] function quantifies the desirability of a given board state. The function is designed to balance various strategic aspects of HexThello, ensuring that the agent makes informed decisions at each step. The evaluation function used in this project is represented as:

$$f(state) = W_1 \times pieces\_diff + W_2 \times valid\_moves + W_3 \times border\_control + W_4 \times bad\_pieces$$

Each term of the function contributes to the estimation of the desirability of the board:

- **Piece Difference** ($pieces\_diff$): This term represents the difference in the number of pieces between the maximizer (in our case maximizer is always the WHITE) and minimizer. A higher piece count is generally favorable, so this term encourages the agent to maximize its pieces while minimizing the opponent's.
- **Mobility** ($valid\_moves$): The number of legal moves available for the agent in a given position. A higher mobility score indicates flexibility and greater control over the board.
- **Border Control** ($border\_control$): This measures the agent's control over border positions on the board. Controlling borders can prevent the opponent from making strong plays and capture agent's pieces.
- **Frontier Pieces** (bad_pieces): Frontier pieces are those adjacent to empty spaces, making them vulnerable to being flipped by the opponent. A high number of frontier pieces is a disadvantage, so this term is weighted negatively.

2.1 Analysis of Evaluation Heuristics.

Some aspects of the evaluation function require fine-tuning:

- **Frontier penalty:** Having many of yours pieces on the board may result to also having more frontier pieces. In this case the frontier heuristic will penalize this position (wich is a good position). To address this problem the weight of the frontier penalty is set to a much smaller value compared to the pieces different weight e.g. $W_1 = 10 \times W_4$.
- During the endgame the agent must prioritize the piece difference over the other heuristics. Therefore, in the last 20-40 moves the agent prioritizes the $piece\_diff$ value. After, a lot of experiments this technic is optimizing the endgame of the agent.

3 Minimax algorithm

A simple implementation of minimax with alpha-beta pruning [1] explores a vast number of board states, leading to slow performance. To address this, two key optimizations are used: **Move Ordering** and **Iterative Deepening Search (IDS)**.

3.1 Move Ordering

Move ordering is a technique that improves the efficiency of the Minimax algorithm. Instead of searching moves in a random order, the agent sorts moves based on evaluation to explore more promising moves first. This allows the pruning mechanism to eliminate irrelevant branches earlier, reducing computation time.

Before exploring possible moves, they are sorted based on heuristic scores:

- **Maximizing player**: Moves are sorted in descending order.
- **Minimizing player**: Moves are sorted in ascending order.

This improves Alpha-Beta Pruning efficiency by allowing more early cut-offs.

3.2 Iterative Deepening Search (IDS)

Iterative Deepening Search (IDS) is a strategy that progressively increases the depth of the Minimax search dynamically, allowing the agent to make better decisions within a set time limit. Instead of searching directly to a fixed depth, IDS works as follows:

1. Start with a shallow depth limit (e.g., d = 2).
2. Increase the depth incrementally (d++).
3. Continue searching until the allotted time limit expires.

IDS ensures that the agent always has a valid move ready, even if time constraints prevent deeper searches. Also, provides the agent with as deep as possible search which results on better estimations of the state and therefore better decisions.

4 Further Expansions and Food for Thought

As we conclude it's worthwhile to reflect on some ideas which could possibly lead to improvements of the agent and are also interesting to investigate:

- Future improvements could implement a phase-based weighting system for the evaluation function. For example, in early game stages, mobility is more valuable, whereas piece count becomes crucial in the late game. If the weights of the evaluation function adjusted dynamically based on the phase of the game, then the agent could be able to take more informed decisions.
- Often the minimax algorithm, as implemented in this project, is visiting (searching) the same nodes (positions) more than one time. We could keep the evaluation of each visited node in a hash map, so we reduce the redundant calculation. This way we trade some memory to have better performance.
- Add more heuristics to the evaluation function. For example, the pieces controlling the center of the board are more powerful because they may offer more mobility later in game.

References

[1] Wikipedia community. *Alpha–beta pruning*. 2024. URL: https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning.

[2] Michael G. Lagoudakis. "Lecture Notes". In: (2024).