COMP 424 Final Project Game: *Reversi Othello*

Student: Marcus Friis Klausen 261241975
Friday Dec 4st, 2024, 11:59PM EST

1. Brief Overview

The strongest approach I found to play Reversi is using Iterative Deepening Minimax combined with Alpha-Beta Pruning and an evaluation function taking into account a weighted current score, mobility and stable pieces for the next move. This strategy tries to balance computational depth with the given time constraint. The method uses:

1. Iterative Deepening to dynamically adjust the search depth based on available time.
2. Alpha-Beta Pruning to reduce the number of nodes evaluated in the search tree, making deeper searches computationally feasible.
3. An evaluation function which considers positional weights, stability, and mobility, reflecting key elements of strong Reversi play.

By prioritizing critical positions like corners and edges while avoiding vulnerable areas, the agent attempts to play strategically to maximize its advantage.
While I think the quality of the agent is fine, as it makes use of non-trivial strategies for evaluation and has reasonable win rates when tested, it could definetly be improved upon, which I will touch on later in the report.

2. Agent Design

2.1 Iterative Deepening Minimax with Alpha-Beta Pruning

As learned in class, making use of iterative deepening progressively increases the depth of the game tree search. Each successive iteration deepens the search by one level while preserving the results of the previous levels. The key advantage is that the algorithm always returns the best evaluated move found up until the timer for making a move runs out.

Alpha-Beta Pruning reduces the effective branching factor by discarding branches that will not affect the final decision. I have tried to implement the algorithm purely based on what was taught in class and from the slides. It maintains two bounds:

1. **Alpha:** The best value the maximizing player can guarantee.
2. **Beta:** The best value the minimizing player can guarantee.

When `beta` $\leq$ `alpha` the algorithm stops exploring deeper in the given branch.
The algorithm makes use of a helper function to sort the valid moves for the current board state to make sure it expands the most strategically promising moves fist. The sorting priority is as follows:

1. Corner moves
2. Edge moves
3. Center moves
4. Moves adjacent to corners

It is important to note that the algorithm assumes that the opponent always chooses the optimal move given the board state, which of course depends on the evaluation function.

2.2 Evaluation Method

The evaluation function evaluates the board state based on 4 heuristics:

1. **Overall Score:** The score of the board state or simply put difference between the amount of player pieces and the amount of opponent pieces.
2. **Positional Advantage:** The positions of a piece has an impact on the quality of that piece, so the pieces on the board have a weight that is factored on the point added to the score of the player with that piece placed. Corners have a heavy positive weight while edge and inner positions have light positive weights. Positions adjacent to corners are given a moderate negative weight, as these positions are vulnerable to be flipped by the opponent.
3. **Mobility:** The mobility of a player is the amount of valid moves a player has given a board state. The higher the mobility the better, and thus higher, the score.
4. **Stability:** The stability of a player is the amount of their played pieces, which cannot be flipped for the rest of the game, meaning that these are secured points till the game is over.

Depending on how far into the game, i.e. how many discs are placed contra how many spaces the board has, the heuristics are weighted differently relative to each other. I have defined 3 stages based on pieces placed p:

1. **Early Game ($p < m^2/2$):** In the early game stability is not considered and mobility is heavily prioritized in favor of positional advantage and player score, as the early game is about creating opportunities for future advantages.
2. **Mid Game ($p < 2m^2/3$):** In the mid game positional advantage and player score is prioritized higher than in the early game, mobility is prioritized lower relative to player score compared to in the early game, while stability is prioritized the highest, as the mid game is about getting a stable point lead going into the late game.
3. **Late Game ($p > m^2 - m$):** In the late game stability is again not considered and player score and positional advantage is very highly prioritized as the game is ending and the agent needs to get as many points possible in as few moves as possible.

3. Quantitative Performance Analysis

In this section I would like to analyze the agent in different aspects of its performance.

3.1 Depth

The depth of the agent has a very high variance. Moves at the beginning of games have a depth in range from 3 to $M-4$ for all board sizes MxM. This is due to the fact that the branching factor depends on the amount of valid moves, which can differ a lot, and thus it can in most cases not get very deep due to the time constraint.

3.2 Breadth

The agent is programmed to evaluate every single valid move up to the given depth for every given board state, so the breadth is thus maximized. Even given the time constraint the agent will always check every possible move up to the depth it iteratively gets to. Thus only the depth has the possibility to be limited.

3.3 Effect of Board Size

The bigger the board size, the more valid moves in the worst case, which means a higher branching factor. The higher the branching factor, the more computations and thus the longer time to get to get deeper into the search. This means that board size is proportional with lower depth in the worst case scenario.

3.4 Heuristics and Move Ordering

For the heuristics I started only using unweighted player score, which performed very poorly. I then added positional advantage, which almost doubled the win rate. I then added the mobility heuristic, which again improved the win rate of the agent significantly. The addition of stability then added a couple extra percent points. All of these were weighted the same for every stage in the game, but weighting them differently proved to be a significant improvement.
For move ordering, I started out just getting the valid moves using the given helper function, but the decision to order the moves in the given priority previously listed gave a significant boost in win rate as well.

3.5 Prediction of Win Rates

For the different opponents, this is how i expect my agent to perform:

1. **Random Agent:** According to my own test, the agent should at win +90% of the time.

2. **Dave:** I have not have time to test my agent against any human player, but I believe it can score wins than losses.

3. **Classmates Agents:** While my agent could be worse, it could certainly use improvements, and thus I think it will place middle of the pack against the other agents created by my classmates.

4. Advantages and Disadvantages of Approach

The approach of my agent has very clear advantages. It ensures that it checks every single valid move to the deepest depth possible given the time limit. This means that no move goes unchecked and the agent always give a move before the time limit is over. It takes advantage of Alpha-Beta pruning to optimize the use of computational resources, resulting in even deeper searches. Furthermore, the evaluation function is dynamic over the course of the game and evaluates based on multiple heuristic, which allows for a more precise evaluation given complex game states.
The agent is not free from disadvantages. It assumes optimal play, and thus if its opponent takes a non optimal move, given the evaluation function, it is possible for the agent to end up in a worse position than anticipated. The agent also generally does not look very deep into the future, as it does not have enough time with the current approach. I also believe the evaluation function could use some work. I created the weights by choosing values I thought to be good, and tuned them a bit afterwards using trial and error based on guesses I made with what strategies I thought made sense.

5. Possible Future Improvements

If I were to introduce improvements to the agent in the future there are a few different things I would try:

- I would definitely improve the evaluation function. There are different ways this could be done, but the best way would probably be to research which strategies are the most important in different stages of the game, and change the heuristic and there weights accordingly,
- I would try to add predefined openings which are most commonly used by the best players, as this would stabilize the early game.
- I would try using MCTS instead of Alpha-Beta pruning. It should be possible to simulate hundreds of thousands of games in the given time constraint, and the fact that the agent would be able to simulate far deeper game states than Alpha-Beta pruning can, could definitely lead to a better agent overall.