COSC 322 AI Project Report

Team 3

Matthew Blanke

Matt Deleeuw

Trevor Gallicano

Dan Schwab

Overview:

Our AI used Alpha-Beta pruning to search for the optimal move on a minimax tree consisting of all possible legal moves we can make, using the current board state as the root node. The value of the moves is determined based on the amount of territory possessed by the player in the resulting game state, compared to the amount that will be possessed by the enemy. This means we are searching for game states in which the most possible spaces are accessible in less moves for us than for the opponent.

Methodology:

Our approach begins by creating a game state tree to search. We begin the process by creating the root node, which stores the current active player (black or white) and the current game state (a 10x10 array of shorts which represents the positions of pieces on the board), and initializing our depth limit to 1. We then create the child nodes for the root by iterating over all possible legal moves that can be made from the root node's game state. Each child node stores both the game state that would result from making that move, and the heuristic value that we calculated for that move (the heuristic function will be covered in detail in the next section.)

Due to the massive branching factor associated with the Game of Amazons, and the time constraint of 30 seconds per move selection, we then remove the children that have a heuristic value below the average (compared to all other children of their parent node), thus removing nodes that are less likely to lead to optimal moves so we can evaluate deeper into the tree on the “best” nodes available. It’s possible that this method could lead to suboptimal moves if we discard a child which ends up leading to our highest guaranteed value, but for the sake of searching as deep into the tree as we can in our limited time window, we’ve found this approach to perform better than the alternative. We then use Alpha-Beta pruning search to determine the highest guaranteed (regardless of opponent’s moves) value node, and choose the move we can make from the current turn that leads to the maximum assured node as the best possible move.

This move is then stored as our current best move, and the process is then repeated after incrementing our depth limit by one, so we generate the next level of the tree and select the best move from that game state tree. This process (next tree level generation, heuristic calculation, discarding low value children, AB pruning, and optimal move selection) is all run on a separate thread from the main program thread (which is listening for server messages and running a timer on our moves), so it will search as deep as it is able to until either it evaluates the entire tree, or the thread is stopped by the main thread when the time limit has been reached. When the timer on the main calling thread reaches its limit, it will send the current best move to the server (after modifying the indexes to match the server’s coordinate system), and stop the move selection thread. We then wait for a response from the server containing the opponents move, apply that move to our current game state, and start building a new search tree with this game state as our root, repeating the process until the game is complete.

Heuristic:

The heuristic we used was the territory heuristic. The value of a game state is calculated by how many tiles are within a players ‘territory’. To compute territory values, we first create a board for each player which contains the minimum amount of moves for the respective player to reach each space on the board (0 if unreachable). A space is in a player’s territory when they require less moves to reach the space than their opponent. This creates a heuristic that seeks to maximize the accessibility of the board for the player while minimizing it for the opponent. We chose this heuristic because it has high performance relative to other heuristics [1], while also being realistic to implement in the time frame that was given.

This heuristic was strong in general, but seemed to undervalue immobilizing opposing queens, which resulted in suboptimal moves in some situations where an opposing queen could have been eliminated from the game. The heuristic was also more effective when the time limit was higher. Lowering the time limit meant that the tree search was not able to get deep enough to evaluate some relatively immediate consequences of moves. As a result, in the early game, sometimes moves would be selected that served to give an immediate gain in territory, but would result in a large loss shortly after.

We considered adding weight to territory values based on the difference in number of minimum moves, and another weighting factor to spaces that are reachable to the player but not the opponent (and vice versa). However, we found no resources to indicate what values would be appropriate for these weights, and we did not have ample time to determine optimal values experimentally.

What to Change in the Future:

While our approach was good for longer decision times, it could have been better for quick decision making. This portion of the report will explore several potential methods of doing so. The main ways that will be described are: using Monte Carlo search tree to pretrain the model, using deep learning to pretrain the model based on previous games, and using deep learning based on the AlphaZero methodology.

Using a pretrained model would have vastly improved the performance in the 20 second decision model. To implement this, we could have used a Monte Carlo search tree, explored a large portion of the tree, and used our heuristic to assign values to each of the nodes. These values could then be saved to a file, and loaded in pre-game. We would only need to do this for the nodes that we believe that another AI would make, as it is unlikely that we would need to account for the opponent making suboptimal moves. Since we would only need to check this smaller portion of moves, it would not take too long to train the AI, doing so on a home PC may take 1-2 days, and the results could be easily saved and transferred, allowing us to quickly evaluate the tree as if we had explored 4-5 layers deep with our iterative deepening.

Using a deep learning model based on previous games would result in a strong AI, but would require a large dataset, and would take a fairly long time to train on a home computer, potentially up to several months. This method could be viable if we had access to extremely high performance computers, or funds to train on the cloud for large amounts of time. This was infeasible for the time and budget of our project though.

Using a deep learning model based on the AlphaZero implementation, the model could have been trained on a home computer in several days. This would have been optimal, but a large technology gap was in the way of learning how to implement this algorithm. This was the biggest roadblock in potentially integrating this into the Java platform that this AI is based in.

In summary, our best option would have been to have quickly train our AI based on our heuristic, and store the results for later reference, allowing us to search deeper into the tree in a short time than our current implementation is able to.

References:

[1] Hensgens, P. P. L. M. "A knowledge-based approach of the game of amazons." *PDF). Universiteit Maastricht, Institute for Knowledge and Agent Technology* (2001).