Technical approach

I implemented the minimax algorithm with alpha-beta pruning, using iterative deepening. This is a recursive approach which looks at a deeper layer each time. At each step, it is determined what the best neighboring state is, based on the heuristics. The alpha and beta values are initialized as the maximum integer value and minimum integer value, respectively. If the player is the maximizing player, the next move is determined by finding the neighboring board state with the highest alpha value. The alpha beta pruning is implemented by breaking from the tree if the alpha found is greater than the beta value. If the player is the minimizing player, the next move is determined by finding the neighboring board state with the lowest beta value. In this case as well, the alpha-beta pruning is implemented by breaking from the tree if the alpha found is greater than the beta value. This alpha-beta pruning is recursively done within the iterative deepening method, so that at each layer the best move is found.

The heuristics I used were number of pieces, distance to the corner, and whether or not the board is a winning or losing state. A board state with fewer of the opponent's pieces is scored higher, by having a factor of 30 multiplied by the number of opponent's pieces subtracted from the board's total score, which I called the payoff. In contrast, a board state with more of the player's own pieces is more highly valued, by having a factor of 60 multiplied by the number of the player's own pieces added to the payoff. If the player is a Swede, a board state with a king closer to the corner will have a higher payoff; this is done by having the payoff add 50, subtracted by the number of tiles to the corner, which will be lower if closer to the corner. If the player is a Muscovite, a board state with the Swede's king closer to the corner will have a lower payoff; this is done by having the payoff subtract 50, subtracted by the number of tiles to the corner. The last heuristic I used is the most obvious, which is whether the board is a winning or losing state. This heuristic is the most highly valued, with a payoff of 50,000 or -50,000 for

winning or losing, respectively. Using the combination of these heuristics, the overall payoff for each board state is assessed.

Motivation

Because tablut is a two-person zero-sum game of perfect information, a game tree of board states can be created in order to describe the moves made by each player at each stage of the game until a winner is determined \citep{viking}. An algorithm which simply searches this tree until the depth of a winning state would be very inefficient, and would surely not be able to perform to the two-second standard of this assignment. For this reason, I chose to implement the minimax search algorithm with alpha-beta pruning, which is O($b^\frac{3m}{4}$) on average, as opposed to O(b^m) for regular minimax \citep{lecture7}. I chose this because I assumed that most students will have heuristics which evaluate similar board states as optimal, and thus minimax will be an efficient way to estimate the behavior of the other player. I chose to implement this using iterative deepening in order to allow search to a specific depth. In this case, I increase the depth by only one. I increase the depth by only one in order to ensure that a move is chosen in time; if I increased it and set a limit for the amount of time spent searching, there is a chance there would be incomplete creation of the board state tree, which may lead to unpredictable game play. In addition, by setting a depth of one, only one “min” or “max” turn is evaluated at once; this leads to more certainty that the opponent will behave in the expected way, while trying to predict their movements multiple times would have have greater uncertainty. Iterative deepening is an ideal way to implement minimax for tablut, a game with a high branching factor and possibly deep terminal nodes, as it is the preferred method for large state spaces, where the solution path length is unknown \citep{lecture2}. In summary: a game tree is created and traversed using iterative deepening with a depth of one, and the minimax algorithm is implemented with alpha-beta pruning.

In order to implement minimax with alpha-beta pruning, there must be an evaluation function. I composed an evaluation function, which I denote as the “payoff” of each board state, which has three components: number of pieces, distance to the corner, and whether or not the board is a winning or losing state. Whether the board is a winning or losing state is the most valuable component of the heuristic, with values of 50,000 and -50,000 respectively. This is so that if there is a winning state on one level, it is sure to be chosen. If this were not the case, there would be the possibility that a board state with a king in a good position and with a high number of pieces would be chosen over a winning state, which would clearly be non-optimal. However, it is also necessary to determine the value of a board state without it being a terminal state. I chose to implement two other heuristics based off the rules of tablut. The other two heuristics have similar importances. A board state with more of the player’s pieces and fewer of the opponent’s pieces will be most highly valued. This is based off extrapolation based on rules 1, 2, and 3: the more there are of an opponent’s pieces, the more possible it is for them to surround the player’s pieces in a way such that they can capture them. It is more important for the player to have more pieces than the opponent to have fewer pieces; this is to take into account that there are cases in which the opponent having more pieces is not necessarily bad, if they are not well placed. The last heuristic is the distance to the closest corner: this is more beneficial if the player is the Swede, as this is, in a way, a measure of how close the player is to winning. In contrast, if the player is the Muscovite, they do not want the player to be close to the player. Using these heuristics, it is possible to capture valuable information about the relative “distance” to a winning state.

Theoretical basis

The goal of the minimax search algorithm is to reduce the time and performance associated with generating all possible moves to find the best move from a given function;

because of this, there has to be an evaluation function. In tablut and other “viking chess” games, these heuristics are not traditionally established, as the games are not played often. However, a crude evaluation function can be determined by weighing certain attributes of the board state. In minimax, each possible move is assigned a value based on this evaluation function. The player making a move is referred to as the MAX player, and the opponent is the MIN player; the goal of the MAX player is to choose moves that provide the highest value while the opponent seeks to minimize this value. When searching to a specific depth, these results propagate, allowing the MAX player to select the best move. A faster minimax algorithm can be implemented using alpha-beta pruning, where there are two variables, alpha and beta, which hold the maximum and minimum values, respectively. This allows branches of the tree to be pruned, so the effective depth and thus the time to find the optimal move is decreased \citep{viking}.

Advantages/disadvantages

The advantages of this method are that it is relatively quick, and it could be easily changed so that iterative deepening looks more moves ahead, which could greatly advantage a player who is sure of the strategy of their opponent. The quickness is due to the implementation of alpha-beta pruning and iterative deepening, which is a “classical time-space tradeoff,” and good for large state spaces \citep{lecture2}. In addition, the high preference for winning and disinclination for losing states ensures that these states are most and least likely to be chosen, which is the optimal behavior. The use of several different heuristics allows for there to be taken into account information about location, likelihood of capture, and whether or not it is a winning or losing state. Overall, this is a relatively quick and comprehensive method of evaluating board states based on various measures of goodness.

There are several disadvantages: reliance on heuristics rely on understanding of a game which is not often played, and the heuristics are not comprehensive. For example, a player

playing as the Swedess may move in a position where capture of the king is possible in order to avoid losing a piece. Using a heuristic function in minimax may also not be the best decision, as minimax assumes that the opponent is playing with the same heuristics. In addition, there are disadvantages to implementation with iterative deepening: it does not take into account the heuristic value when deciding which nodes to deepen – it is an uninformed search method. If the allowed depth were more than 1, although it would be complete, it would not be optimal. In contrast, it could be improved by using iterative deepening A*, which would use a heuristic-value limit rather than a depth limit. This could be further improved by keeping board states which have already been expanded in memory, using the algorithm known as SMA* \citep{lecture3}. In general, the disadvantages are caused by reliance on a concrete evaluation function.

Possible improvements

The main difficulty with developing a player of tablut is developing a comprehensive evaluation function, which is better than the evaluation function of the opponents. Two ways to avoid setting immutable heuristics are to run Monte Carlo Tree Search (MCTS) \citep{viking}, or to use an evolutionary algorithm \citep{hnefatafl}. MCTS starts with a game tree consisting of only a root node and progressively builds up a game tree in memory using the results of past simulations, becoming increasingly better at estimating the values of the most favorable nodes \citep{viking}. As such, creation of a concrete evaluation function is avoided. An evolutionary algorithm acts similarly in favoring, through selection, players that have discovered general principles of good play \citep{hnefatafl}. However, evolutionary methods are plagued by problems such as stagnation and cycling, and population size can be limited by the high branching factor of games such as tablut \citep{hnefatafl}. Thus, my preferred method for moving forward would be to implement MCTS.