Report

Claudine LeBosquain 260449697

Approach and Motivation

The player is comprised of two main segments, as well as various helper method. The first of these segments involves building a tree, up to a set depth, that collects all the legal moves that one can take during a turn. With this tree I would be able to search for the best attainable board state given any number of hops, instead of looking only one hop at a time (this is what getLegalMoves() returns). The tree is comprised of custom, self-made nodes, and a general tree class that I took from the internet which aided my organization[1].

The second segment involves using a heuristic function to evaluate which node has the best board state. The heuristic function performs breadth first search (skeleton code was obtained on the internet and changed to fit my specifications[2]), and at each node it applies the evaluation condition. It keeps track of the best node and returns it at the end of the function. There are three aspects to the evaluation condition, and weights are assigned to each of these aspects. These weights differ based on the number of turns that have elapsed. The three aspects to the evaluation condition are:

1. Distance Function I used a standard Manhattan Distance calculation between each of my player's pieces and the square that is farthest away in the goal state. I attempt to *minimize* the distance function at each board state. The distance function is considered for all turns.
2. Mini-Chains This method returns whether or not a "mini-chain" is created, by looking at the arrangement of pieces around each of the player's pieces. The more mini-chains one has, the better a position one is in, as chains can be used

[1] http://stackoverflow.com/questions/3522454/java-SearchLegalMovesTree-data-structure
[2] http://www.codeproject.com/Articles/32212/Introduction-to-Graph-with-Breadth-First-Search-BF

for players to hop back and forth and shuttle their pieces across the board. I attempt to *maximize* the number of mini-chains for my partner and I (for all turns less than 40), while *minimizing* the number of mini-chains for my opponents (only for turns 25-40).

3. Pieces In Base This method simply counts the number of pieces remaining in the base of the player. Have pieces in base may result in disqualification, or trapping either intentionally (by your opponent) or unintentionally (by your ally, trying to move in). I try to *minimize* the number of pieces remaining in my base. This is only considered for turns less than 25.

Finally, a helper function follows the parent trace backward, creating a movesList which contains all of the moves the player needs to attain this board state.

Theoretical Basis

George Bell wrote an excellent paper[3] in which he describes three phases of game play: the gambit (opening moves), the melee (army interaction), and the packing (attempting to get into goal as quickly as possible). I loosely modelled my algorithm after this idea. The algorithm sections which correspond to these phases are :

- Less than 25 turns, which corresponds to the gambit. In this phase, there is a large weight placed on getting your pieces out of base, as well as setting up chains for yourself.
- Less than 40 turns but more than 25 turns, which corresponds to the melee. In this phase, there is an emphasis on creating chains for yourself and blocking chains of others.

[3] Bell, George. "The Shortest Game of Chinese Checkers and Related Problems." *Electronic Journal of Combinatorial Number Theory*. 9. (2009): n. page. Web. 11 Apr. 2014. <http://arxiv.org/pdf/0803.1245.pdf>.

- Greater than 40 turns. In this phase, the only driver is manhattan distance. By following the manhattan distance, the heuristic should drive pieces to enter goal as quickly as possible.

Bell also discusses chains, and “ladders” for pieces to hop along. This was my the theoretical basis behind the mini-chains method.

Advantages and Disadvantages

The clear advantage of my player is that it is simple and easy to understand. It relies only knowledge of simple data structures, and a clever heuristic function. It is also very fast, as it must be to avoid timing out. Another advantage is that it is very flexible. In the evaluation method, there are many weights that can be adjusted depending on the situation (whether one wants to play more aggressive, have greater focus on getting to the goal, or set up more advanced chains). One can also adjust the depth of the legalMoves tree, which allows you accommodate various time constraints.

There are also obvious disadvantages to my player. First and foremost is the fact that my player is greedy, and does not “look ahead” to see what possible moves might be better in the future - instead it picks the move that looks best for this turn. Additionally, the legal moves tree is only calculated up to a fixed depth. This prevents it from looking at all possible hops for a move and instead only returns a partial moves list.

I expect my player to be especially weak against players who attempt to trap their opponents in their base early on. There is a brief period at the beginning where my player is susceptible to being locked inside it’s base, due to the fact that the motivating factor to leave the base is based on the number of pieces in base, and the

piece at the very back of the base will not see a reduction in its heuristic value unless it can hop out of base entirely.

Another of my players weaknesses is that it will occasionally choose to preferentially build mini chains even if the chain is not useful (for example, if all your pieces have already been transported to the goal side of the mini chain). This might prevent the player for entering goal sooner. Finally, if the weight on mini chains is too high, occasionally my player will not enter goal

Other Approaches and Possible Improvements

The first thing I attempted was a naive manhattan distance. This accomplished the goal of moving my pieces into the goal zone, but is far inferior to my current player. Even the next "model", which used a heuristic involving mini-chains similar to my current player, but which did not have a tree of legal-moves, and so was only able to calculate one hop at a time. My current player consistently bests this other player.

Early on, I had also attempted to implement a basic minimax tree. However, I was not able to look very many moves ahead before the size of the tree caused my player to time out. Even after disabling the "time-out" feature, I found that the more moves I looked ahead (using the minimax tree), the worse my player performed. After discussing with some of my colleagues who had also experienced the same result, we came to the realization that minimax assumes that the opponent is not trying to minimize your outcome (as minimax assumes of adversarial opponents), but rather they are trying to maximize their own gains.

Thus, to improve my player, I would have needed to implement some sort of "maximax" algorithm, preferably with some sort of pruning, wherein all players were trying to maximize their returns. However, due to time constraints, this was not

feasible, and I simply deleted my minimax tree in favor of the naive algorithm which performed better. Implementing Monte-Carlo roll-outs along with the proposed "maximax" tree would also be good, as they would allow me to look a greater number of moves ahead while keeping my player within the time limit.

I would like to have had more time to investigate the weights of my evaluation methods, and the number of turns I spend in each phase. Perhaps one way of improving my player would be to engage in self-play over a range of values for the weights which would allow me to determine the ideal set. Moreover, perhaps there are different sets of weights which are idea for different play styles. I would investigate this as a possible improvement. Potentially, some sort of simplistic learning algorithm might be learn to know how long (how many turns) to spend in each phase. It might also be interesting to try hard-coding some of the first moves, as this was a technique that I did not utilize in my player, but that seems to yield positive results for others.

Finally, it might be interesting to try implement some of the probabilistic methods that we talked about in class. They might be too difficult to code in the short term, but perhaps given more time, a probabilistic approach to setting the weights, or a simulated annealing sort of approach, where the temperature was picked from a distribution based on time, might allow me to improve my AI.