George Macrae

260401458

Comp 424 Artificial Intelligence

Project

1. An explanation of how your program works, and a motivation for your approach.

The George Macrae s260401458 make-uniform-choices-home-without-waste halma artificial intelligence agent (widely renowned as the ‘MUCH-WOW halma AI’) is a fully functional halma player motivated by the desire to embarrass opponents when they allow their adversary make large chains of moves (jumps). MUCH-WOW begins it’s turn by making a detailed calculation of the the most advantageous move by comparing the results of the evaluation function on the current state of the board with the state of the board after each completed turn, which could be a string of jumps. If a string of jumps is made, then each move is stored into an array. MUCH-WOW checks if that array is empty at the start of each move, popping a turn if its not empty or filling the array using the algorithm described above and then popping the array for a move, allowing MUCH-WOW to run the algorithm once per turn.

The evaluation function (MyTools.getManhat(...)) is a poetically simple yet highly effective attestment to complex brilliance behind the game of halma. MUCH-WOW uses a finely tuned and slightly altered heuristic widely known manhattan distance (as described in class). The bases of each player are hard coded into arrays in layers. The first layer being the corner coordinate, the 2nd layer being the 3 spots surrounding the corner and so on (ie player 3 has the base { (15,15),(14,15),(14,14),(15,14)...}. When the evaluation function is performed, the

distance of each piece to the furthest spot in the goal base that is unoccupied or is occupide by an enemy piece, is summed. The lower the result of the evaluation function, the better.

To make sure pieces are moved out of the base before 100 turns, after 80 moves, MUCH-WOW greedily picks moves that will move pieces that are still within the base until there are no pieces left in the base. 80 moves was chosen such that MUCH-WOW would first be able to get pieces into the middle, allowing it to seize dominance over unworthy adversaries.

Near the end of the game, the MUCH-WOW agent ran into some problems as no greedy jumping moves would be advantageous to the the manhattan distance leaving the moves array empty. This was solved by using a simpler approach which again looked at all the moves; if the move was from the goal zone then it was skipped , if the move was to the goal zone it was made. Otherwise, instead of looking at all the pieces combined distances, the move which reduced any one piece's distance to a goal was taken. And if the best move tied with another (which was the main issue as the moves score would tie with the current score) then one of these moves was chosen at random as long as the move was not to the origin of the last move that was made with that piece.

2. A brief description of the theoretical basis of the approach (about a half-page, in most cases);

At first MUCH-WOW was designed to be a recursive algorithm which would delve deep into the tree of states (this can still be simulated by altering the DEPTH constant). However, the enormous branching factor prevented MUCH-WOW from completing this with efficiency which

will be discussed later. MUCH-WOW now relies heavily on the evaluation function and that there will be many opportunities to make several jumps in a single turn. MUCH-WOW is very much a greedy AI agent as it only considers the moves at present and isn't concerned with the future.

3. A summary of the advantages and disadvantages of your approach, expected failure modes, or weaknesses of your program.

Since MUCH-WOW is a greedy agent there are some critical weaknesses. MUCH-WOW does not look at future moves and therefore could be sacrificing a more optimal play out for an immediate advantage. Although, in some cases this could be advantageous (for seizing control of the mid game for instance) MUCH-WOW suffers in the end game when many moves result in the same manhattan distance difference ,or when jumping ladders are broken because a piece in the jump chain has an opportunity to jump, leaving pieces behind stranded.

Another weakness of MUCH-WOW could be its heuristic. For most aspects of the game the modified heuristic is a very powerful one but can also be responsible for trailing pieces and not forming chains.

Yet one more weakness of MUCH-WOW is when a jump is made , not all moves following that jump (ie. moves following the first move of a turn) are considered, only the best one. This is a weakness since a jump in a sub-optimal direction might lead to several jumps in a optimal direction.

4. If you tried other approaches during the course of the project, summarize them briefly

and discuss how they compared to your final approach.

As mentioned, MUCH-WOW was designed to be a recursive algorithm with alpha beta pruning, however the branching factor was far too large for this to be implemented effectively. Since a depth of 4 (only a depth of 2 could be tested in the amount of time allowed per move) needed to be reached to achieve simulation of MUCH-WOW's next move , the search was ineffective.

There was also an earlier version of MUCH-WOW (super-uniform-choices-home-without-waiting halma agent or SUCH-WOW for short), which did a recursive search but did not consider all moves (randomly removing moves until X amount remained). Again, this attempt was futile since in the mid game there can be over a hundred moves and only about 60 could be expanded which meant that the optimal move was often omitted.

5. A brief description (max. half page) of how you would go about improving your player (e.g. by introducing other AI techniques, changing internal representation etc.).

The most obvious change to MUCH-WOW would be to improve the way it finishes simulating turns. Since MUCH-WOW looks at all moves and then greedily picks the best jumps after the move, it might not be picking the optimal jumps (if, for example, it has to make a jump backwards which would lead to several jump forwards).

MUCH-WOW could also be improved by implementing increasing the allotted time for a move since it can search deeper for more optimal moves using the techniques described previously.

Instead of searching deeper, a better heuristic could be used to give better scores that set up MUCH-WOW for long jumping chains. There would need to be balance such that MUCH-WOW doesn't spend all its moves setting up rather than playing moves that move it forward.