Problem Description

The purpose of the project was to implement A.I. which uses Minimax and Alpha-Beta pruning for the game "The Mad King!" which is a fully observable deterministic game played by 2 players on a 5x5 grid. Player 1 has 3 guards and 1 king, which can all move vertically or horizontally, the king can jump guards onto an unoccupied state, can attack one of player 2's pieces by having 2 of it's own pieces adjacent to player 2's piece and moving onto the space player 2 occupies, and wins by moving the king to row 4 of the board. Player 2 has 5 dragons which can move one space in any direction, can turn a guard into a dragon by surrounding it with 3 dragons, and wins by surrounding the king by 3 dragons in a position where the king cannot move. Players trade turns and must make a move if a move can be made.

Solution Description

On the turn of the player operated by the computer, the states that can be reached on the board through a single move made by one of its players are examined. For each state examined, the states which can be reached through one move from the opponent are also examined. For each state considered that the opponent can make, each move the player can make after that is also considered. This is recursively done for new states to be considered creating a tree of states that can be reached with a depth of 8 moves from the root(which is the current state of the game) and a depth of 10 moves for states which look better than the rest.

Minimax Search

The player operated by the computer chose to make a move which was predicted to be the best option for it based on the leaves of the tree created through its recursive state search. The best states are determined through a depth first search in which it assumes that the opponent will be trying to win. So for every move the computer player can make, it assumes the opponent will choose an optimal move. The move chosen by the computer then is that which may reach the optimal game state of all the leaves from the tree created from recursive state search, assuming that its opponent will be playing to win. The optimal states are determined by arbitrary values that were assigned to certain conditions such as the number of remaining dragons, remaining guards, how far the king is from where it needs to be, and whether the king is captured. Every leaf node in the tree receives a value based on these assignments. Parent nodes receive the maximum of all the values of their children if the children can be reached from a move made by the computer, and receive the minimum value of all their children if the children are reached by a move made from the computer players opponent. The root node chooses a move which results in the state with the maximum value of all its children, because all the child nodes of the root are reached through a move made by the computer.

Alpha Beta Pruning

Because the opponent in the game will be playing to win, the algorithm for the computer player is written such that if a move made by the computer results in a state in which its opponent can reach a more optimal state for him or herself, it is assumed the opponent will make the moves that reach this state. While the minimax algorithm is running, nodes reachable by a move made by the computer player are assigned a temporary minimum value until it is determined by the minimax algorithm that the maximum value is otherwise. If the temporary value of a node reachable from the computer is already lower than the minimum value of one of the nodes siblings, the search through all the descendants of this node stops. This is because minimum values can only decrease and the parent of this node will select the maximum value from all its children. If the current node being looked at has a lower value than a sibling, then its parent will not choose it, so it is pointless to consider the search. For nodes reachable from the computers opponent, if it's temporary maximum value is already higher than one of its siblings, the search into this nodes descendants stops, because the parent of this node chooses the minimum value of all its children.

Our program implements this by passing the maximum value found so far from all of a nodes siblings (known as the alpha value) to the search performed on each of its children for nodes reachable from the computer. If the children receive a temporary value lower than this alpha value received, no more searches are called to the descendants of this node. This is because the nodes parent receives the maximum of all its children while the nodes temporary minimum value can only become lower.

Nodes reachable from the computers opponent are passed the minimum value of all their siblings (known as the beta value) and if these nodes receive a temporary value (which is the maximum of all its children) which is higher than this beta value, the search to the descendants of this node stops. This is because the nodes parent receives the minimum of all its children while the nodes temporary maximum value can only become higher.

Implementation Description

Utility Assignment

We determined the value of each state based on a utility function which assigns a score of 1 for every guard left, 1 for every dragon captured, 20 for each row forward the king is in the board, -1 for every dragon left, -1 for every guard captured and -20 for the capture of the king. Each leaf node then receives score which is reflective of how good the state is and every other node receives the maximum or minimum value of its children depending on whether the node is

reachable from the computer player or the computers opponent, as explained in the solution description.

Selective Deepening

For a more accurate search, selective deepening was used on the search tree every node of depth 8 deepens the search to a depth of 10 when the score of these nodes is significant (>35 in a search for the maximum, and <-35 in a search for the minimum). The search is stopped by the base case of the minimax or alpha beta pruning algorithm, which checks if a depth of 8 has been reached with a non-significant score, or a depth of 10 with a significant score.

Minimax Technicalities

The maximum value for a node(state) is determined through a function which calls a minimum value helper function for the nodes children . The minimum value helper function calls a maximum value helper function for all its children which calls the minimum value helper function for the children of all the nodes passed into it. This is continued until the desired depth is reached when the utility value assigned to the state becomes the return value of the function.

Alpha-Beta Pruning Technicalities

The alpha-beta algorithm is implemented using an algorithm very similar to the minimax algorithm with the differences in indicated in the solutions description. Values of Alpha and Beta are initially assigned to be -∞ and ∞ respectively in the maxValue and minValue functions. In a for loop through all the child nodes of a state, the alpha value is continually assigned the max value of all its children for states reachable from the computer players opponent and the beta value is continually assigned to the lowest value of all its children for states reachable from the a move made by the computer player. A conditional within the for loop which searches through all of a nodes children stops the loop and immediately returns the current maximum value which are greater than or equal to the beta value, and minimum values which are less than or equal to the alpha value.

Increasing Alpha-Beta Pruning's Effectiveness Through Sort

We attempted to increase the amount of states excluded from search due to alpha beta pruning by sorting child states according to what seems the best currently. When these states are sorted from worst to best, the likely worst state of all the child nodes will likely be found first then. When looking for the best move for the opponent of the computer player then, values which are higher than the beta value(which the opponent would not let the computer pick) will be recognized and more nodes can be excluded from the search. When sorting states reachable from a move made by the computer, if the states are sorted from best to worst, then alpha will initially receive a very high value and more children will be excluded from search because lower value states reachable from the opponent will be seen earlier.

Beam Search

To quicken the search for the best move, a type of beam search was implemented. This involved reducing the amount of children searched once a depth of 7 was reached in the tree. When this depth was reached only the 3 highest utility child states are searched recursively, although the utility of all the child states in still determined, these child nodes are not searched recursively which reduces the number of branches in the search tree. Because the states are already sorted by which ones are expected to be the best, the 3 best states are simply taken from this sorted list.

Transposition Table

A table was implanted into the A.I.'s search algorithm which kept track of the moves which the computer made. If the game was in a state which it had already been in, the computer checked the table for what move had been previously used in this state, and made the same move.

Results

The two co-creators of the A.I. in this project played games of the mad king trying to win against it. In the tables below, one of the creators is named player 1 while the other is named player 2.

Player 1 played 6 games with our minimax algorithm enabled with no alpha beta pruning, 6 with alpha beta pruning, 6 with alpha beta pruning and selective deepening, and 6 with alpha-beta pruning with beam search enabled. Half the games were played as the dragons and half the games were played as Kings and Guards.

Player 2 played 6 games with our minimax algorithm enabled with no alpha beta pruning, 6 with alpha beta pruning with beam search enabled, 6 with alpha beta pruning with selective deepening enabled, and 6 with alpha beta pruning with both beam search and selective deepening enabled.

The tables below represent the results of the matches with a win representing a win for the A.I.

We searched using a deeper depth when the algorithm used enabled a deeper search to be able to happen without increasing the time very much.

Computer(Dragons) vs Player 1(Kings and Guards)

Search	Wins	Losses	Draws
Minimax	2	0	1
Alpha-Beta Pruning	2	0	1
Alpha-Beta Pruning with Selective Deepening	2	0	1
Alpha-Beta Pruning with Beam Search	2	0	1

Computer(Kings and Guards) vs Player 1(Dragons)

Search	Wins	Losses	Draws
Minimax	0	3	0
Alpha-Beta Pruning	2	0	1

Alpha-Beta Pruning with Selective Deepening	1	1	1
Alpha-Beta Pruning with Beam Search	1	1	1

Computer(Dragons) vs Player 2(Kings and Guards)

Search	Wins	Losses	Draws
Minimax	0	3	0
Alpha-Beta Pruning	2	0	1
Alpha-Beta Pruning with Selective Deepening Disabled	1	0	2
Alpha-Beta Pruning with Beam Search Disabled	2	0	2

Computer(Kings and Guards) vs Player 2(Dragons)

Search	Wins	Losses	Draws
Minimax	0	3	0
Alpha-Beta Pruning	3	0	0
Alpha-Beta Pruning with Selective Deepening Disabled	3	0	0
Alpha-Beta Pruning with Beam Search Disabled	2	0	1

Time Measurements

Search	Depth	Total Time (s)
Alpha Beta Pruning	8	808.508
Alpha Beta Pruning with Transposition table	8	734.577
Alpha Beta Pruning with Selective Deepening	10	951.597
Alpha Beta Pruning with Beam Search	12	934.038

Conclusion

Without alpha-beta pruning, the search can not go deep enough in a reasonable amount of time to make a competitive A.I. With only the minimax enabled, the computer takes a long time to take a move, too long for the competitor to wait. When the depth of the search tree is not very deep, it is difficult for the computer to make good long term moves and becomes easy for the competitor to find strategies to beat the computer because the computer could only plan a few moves ahead, while making use of an imperfect utility function.

I found selective deepening to be important in making the A.I. nearly unbeatable. In the games where selective deepening was enabled, it was very hard to plan a game in which the A.I. would lose. Whether or not beam search was enabled seemed to have little effect on the moves which the A.I. made so long as selective deepening was enabled. When selective deepening was used without beam search, it took a tremendous amount of time for the computer to make a move when the computer was in a near winning position. This is because when the computer had almost won, almost every move it could make would be considered very high utility from the utility function, and thus almost every branch would be searched deeper. When the beam search and selective deepening were used in combination, the computer was able to make really good moves while also operating at a more reasonable speed, although it still took the computer a long time to make a move. Clearly alpha-beta pruning, and appropriate heuristics, are needed in this game to secure more wins for the computer.

The transposition table had little effect on the game, unless a position was reached where the two players were making the same moves repeatedly, in which case the computer would make an almost instantaneous move.

The player playing the king can usually end the game in at least a draw by backing into the corner. This is because in this position the king cannot be surrounded by three guards. Other than this the game is close to evenly match. With an effective A.I. it becomes nearly impossible to beat this game with the king because the dragons will stretch across the board and not let anyone pass. With two computers playing against each-other, the result is usually a draw, because neither side is willing to open a weak spot for the other side to take advantage of. I observed a few games that ended in both computers making the same move over and over again because the best result which could be observed was the previous position the piece was in. Each game took on average 27 moves, so an A.I. which could efficiently look about this many moves into the future would have next to no chance of losing. There is little chance of an effective algorithm which could do this however because in each state there is roughly 10 moves a player can make, making roughly 10^{27} states to look through to find a winning state in the game and this is just too many states to look through.