Artificial Intelligence

Assignment # 1

By: Gary Corcoran

Given the three states before, I created a recursive minimax function. This function recursively visits each node in a depth-first manner. This means is started at each of the leaf nodes and as the recursion unravels, returns the maximum or minimum value of the children nodes depending on if it is the AI agent's turn, or its opponent's turn.

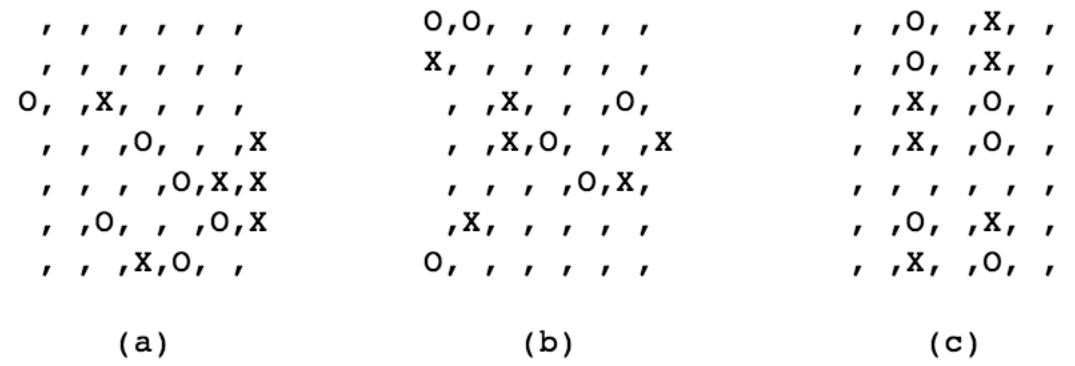

Figure 1 – Three initial states for minimax and alpha beta pruning game trees

Unfortunately for this many possible permutations, the game tree consisting of all possible nodes would impossible to incorporate. Thus, it required to program depth-limited search. This algorithm proved to be a lot more effective in cutting down the state space to a feasible amount. This algorithm was tested using three different limits for the depth-limited search: 3, 4, and 5. For a depth of 3, the game tree for figure 1 (a) consisted of 1591 nodes. See figure 2 below. Thus, all 1591 nodes had to be explored for minimax algorithm. For a depth of 4, the game tree consisted of 15'493 nodes, see figure 3 below.

```
 1 2 3 4 5 6 7
1 , , , , , ,
2 , , , , , ,
30, ,X, , , ,
4 , , ,O, , ,X
5 , , , ,O,X,X
6 , ,O, , ,O,
7 , , ,X,O, ,X
Action = (6, 6, 1140).

 1 2 3 4 5 6 7
1 , , , , , ,
2 , , , , , ,
30, ,X, , , ,
4 , , ,O, , ,X
5 , , , ,O,X,X
6 , , , , , ,
7 , ,O,X,O,O,X
Action = (7, 3, 20).

Totals nodes in game tree = 1591
```

Figure 2 – Total nodes explored for minimax (depth 3)

```
 1 2 3 4 5 6 7
1 , , , , , ,
2 , , , , , ,
30, ,X, , , ,
4 , , ,O, , ,X
5 , , , ,O, ,X
6 , , , , ,X,
7 , ,O,X,O,O,X
Action = (6, 6, 100).

 1 2 3 4 5 6 7
1 , , , , , ,
2 , , , , , ,
30, ,X, , , ,
4 , , ,O, , ,X
5 , , , ,O,X,X
6 , , , , , ,X
7 , ,O,X,O,O,
Action = (6, 7, 100).

Totals nodes in game tree = 15493
```

Figure 3 – Total nodes explored for minimax (depth 5)

Lastly, for a depth limit of 5, the total number of nodes explored for minimax is 196'926. See figure 4 below.

```
 1 2 3 4 5 6 7
1 , , , , , ,
2 , , , , , ,
30, ,X, , , ,
4 , , ,0, , ,X
5 , , , ,0,X,X
6 , , , , , ,X
7 ,0, ,X,0,0,
Action = (7, 2, 140).

 1 2 3 4 5 6 7
1 , , , , , ,
2 , , , , , ,
30, ,X, , , ,
4 , , ,0, , ,X
5 , , , ,0,X,X
6 , , , , , ,X
7 , ,0,X,0, ,0
Action = (7, 7, -280).

Totals nodes in game tree = 196923
```

Figure 4 – Total nodes explore for minimax (depth 5)

A similar process was carried out for the two remaining initial states, figure 1 b) and c). The table below shows the break down.

	Figure 1 a)	Figure 1 b)	Figure 1 c)
Depth 3	1’591 nodes	2’452 nodes	2’949 nodes
Depth 4	15’493 nodes	34’039 nodes	40’404 nodes
Depth 5	196’923 nodes	434’757 nodes	556’422 nodes

Table 1 – Number of nodes explored for minimax

Alpha beta pruning allows one to cut down the number of numbers by pruning nodes where alpha <= beta. Luckily, this means achieve the same result by visiting fewer nodes in the game tree. The table below shows the total number of nodes explored for each of the initial states in figure 1.

	Figure 1 a)	Figure 1 b)	Figure 1 c)
Depth 3	831 nodes	1’304 nodes	1’443 nodes
Depth 4	5’341 nodes	15’125 nodes	20’042 nodes
Depth 5	20’959 nodes	48’493 nodes	59’913 nodes

Table 2 – Number of nodes explored for alpha beta pruning

Another effective approach is to use a heuristic evaluation function for non-terminal nodes. In the previous algorithms described, I used the zero-sum rule of assigning a +1 to a win for the AI, -1 for a human win, and 0 for neither. However, instead of these trivial assignments, you can use a heuristic evaluation function. Table 3 below shows the results of this method.

	Figure 1 a)	Figure 1 b)	Figure 1 c)
Depth 3	526 nodes	1'253 nodes	443 nodes
Depth 4	1'437 nodes	5'102 nodes	1'984 nodes
Depth 5	15'417 nodes	31'655 nodes	13'913 nodes

The number of states can be decreased again if you evaluate the states in a particular way. If you visit the "best" states first, you can make alpha beta pruning a lot more effective and thus means you can explore even less nodes. In order to do this, I created an iterative deepening algorithm. At each iteration the algorithm calculates the best move, from alpha beta pruning, and uses this move for the first move for the next iteration. This process proved to cut down the amount of nodes more than I expected. The table below shows the number of nodes explored using this technique. Another option I looked into, was instead of using iterative deepening, was to use the

	Figure 1 a)	Figure 1 b)	Figure 1 c)
Depth 3	210 nodes	216 nodes	371 nodes
Depth 4	460 nodes	576 nodes	1'400 nodes
Depth 5	5'609 nodes	2'831 nodes	5'942 nodes

Table 3 – Number of nodes explored for alpha beta pruning using heuristic evaluation function

The heuristic function I decided to use was one I made while playing against the AI agent I programmed and I continued to fine turn with arbitrary constant multipliers. Firstly, my heuristic looks at the distance between each AI's pieces on the board. This algorithm counts the distance between each piece and removes this from the total score. The next aspects I look at are clusters of groups. If the game board has a cluster of two or more AI's pieces, a constant is added to the total score for the board. The same thing happens for the human's pieces, however, this is subtracted from the score. Lastly, I look at if a move creates a gap between two adjacent human's pieces. If so, then this means the human player can connect two or more pieces and potentially win. This removes a

constant from the overall score of the board. Using these functions I created a score for each game board.

However, as you might guess, this heuristic takes awhile to calculate, since it needs to iterator through the game board many times. However, using this heuristic still allowed me to cut back the number of nodes being explored. In order for me to do this, I had to try to make the evaluation function as efficient as possible so I spend a lot of time on this part to try to fine tune it so it would not iterator through the game board more times than it had to. When I first started using this evaluation function, it was not very efficient so it actually made my program not be able to go as deep in the game tree as it did without using the function.

Overall, I spend a lot of time on the coding portion of this assignment, as I got tunnel vision a bit. However, I learned a lot from this assignment and trying to implement each of the algorithms and seeing how they can cut down the number of nodes explored at each point.