ECSE 526: Assignment #1

Presented to: Jeremy Cooperstock

Prepared by

Rahul Behal [260773885]

[ECSE 526 – Artificial Intellgience]

Department of Electrical and Computer Engineering

McGill University

2020.09.22

Table of Contents

1 Part 1 3

Question 1 3

Question 2 5

Question 3 8

2 Part 2 9

Question 1 9

Question 2 11

Question 3 12

Question 4 14

3 References 18

1 Part 1

Question 1

Indicate the total number of states visited by your program, starting from each state shown, when using depth cutoffs of 3, 4, 5 and 6, both with minimax and alpha-beta. Assume it is white's turn to play.

The minimax and alphabeta algorithms were both run on three different initial game states; each was run with a depth cutoff of 3, 4, 5, and 6. The pseudocode from Russell & Norvig [1] were referemced in the coding of these algorithms. The starting player will also be white, as given in the problem statement. The 7x7 gameboard was represented by a 2D array and the initial gamestates can be seen in Figure 1.

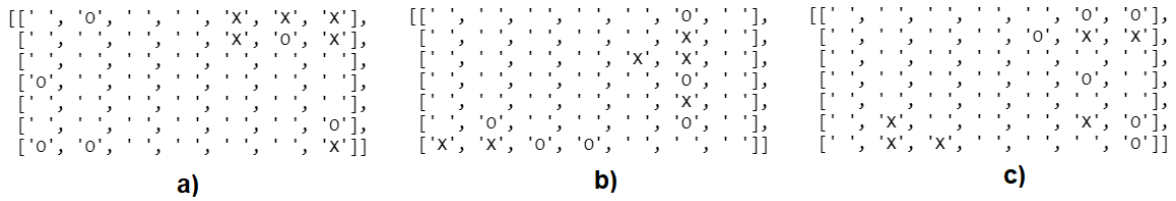

Figure 1 Initial game states

The number of unique game states visited by the algorithms will be calculated for each depth cutoff and initial game state. The results can be seen graphically below.

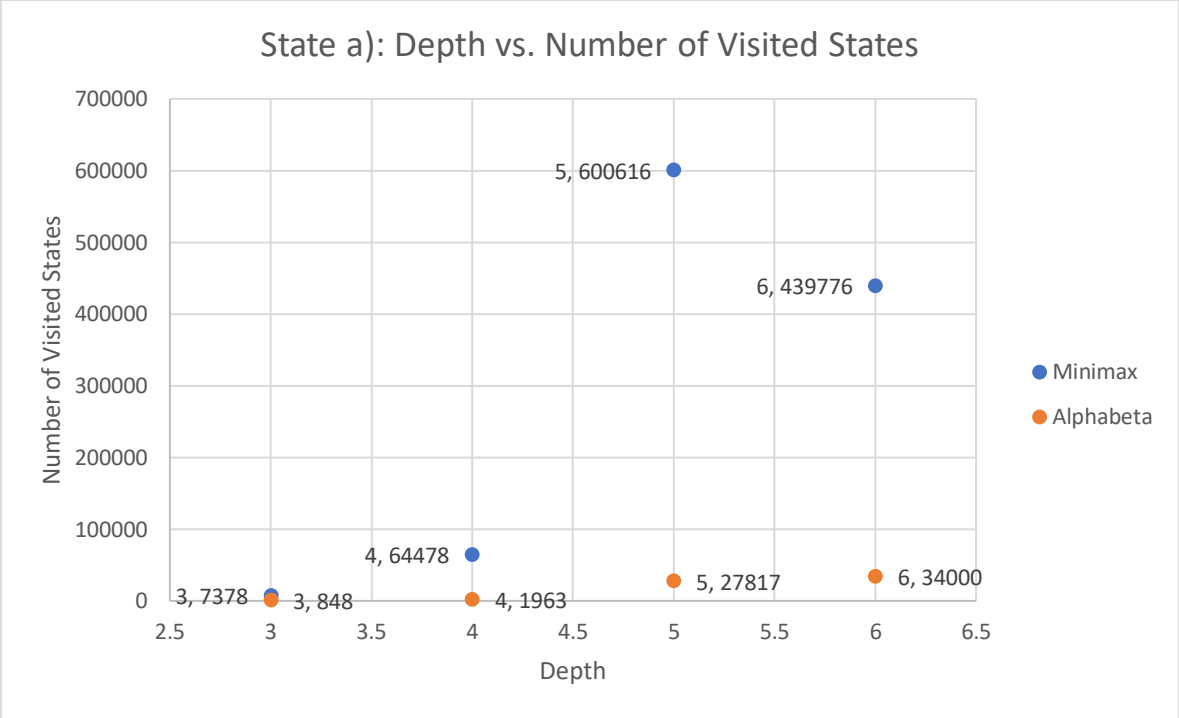

Figure 2 State a): Depth vs Number of Visited States

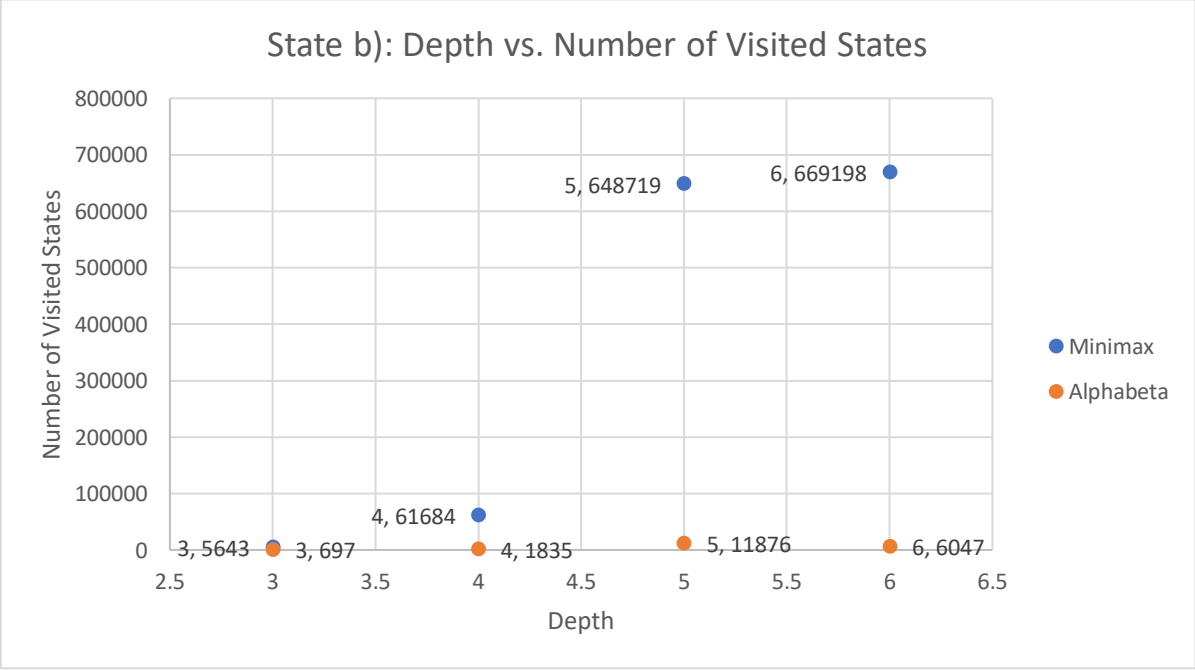

Figure 3 State b): Depth vs Number of Visited States

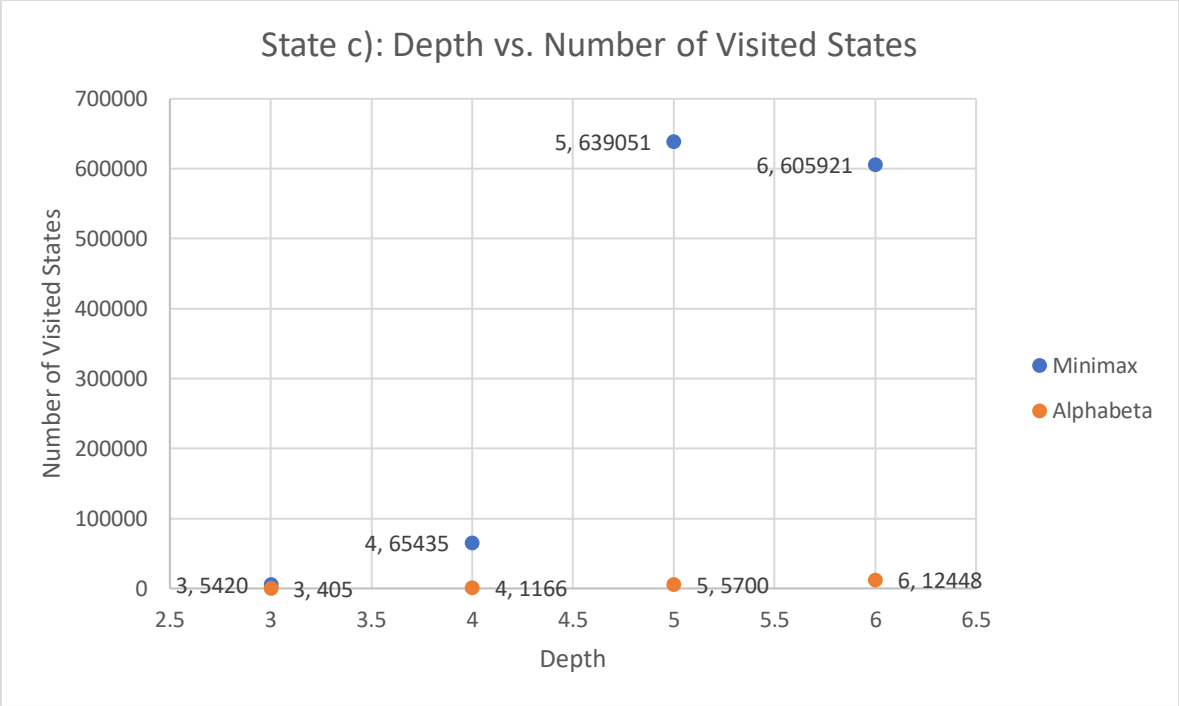

Figure 4 State c): Depth vs Number of Visited States

Question 2

Does the number of states visited depend on the order in which you generate new states during the search? 1) Explain your answer and 2) justify it using results from your program.

For the minimax algorithm, the number of states visited does not depend on the order in which you generate new states during the search. This is because minimax explores and evaluates all possible states until the depth cutoff or if the state is a terminal one. No matter how the moves are ordered during the search, minimax has to go through all possible actions and thus all possible game states.

For alphabeta pruning, the number of states visited does depend on the order in which you generate new states during the search. This is because the alphabeta algorithm prunes the branch if it evaluates to be worse than a move it has already evaluated. As such, if it evaluates better moves sooner, the number of states traversed will be significantly less since many states will be pruned away. Clearly the order in which you generate new states affects the total number of states visited here.

In order to justify this claim, move ordering can be switched and the resulting effect on number of states visited can be showcased. If the above logic is correct, there would be no change in the number of states visited for the minimax algorithm, however the number would change for the alphabeta algorithm.

Currently moves are being generated top to bottom. Through adding the piece of code in Figure 5 at the end of the *get_all_possible_moves()* function that generates new states, move ordering will become random and the difference in number of visited states can be viewed graphically. A log scale will also be used for number of states visited for better visibility.

```
random.shuffle(possible_moves)
return possible_moves
```

Figure 5 Changing the order of searching states

For simplicities sake, the algorithms will be compared with the same initial conditions as example c) from Question 1.

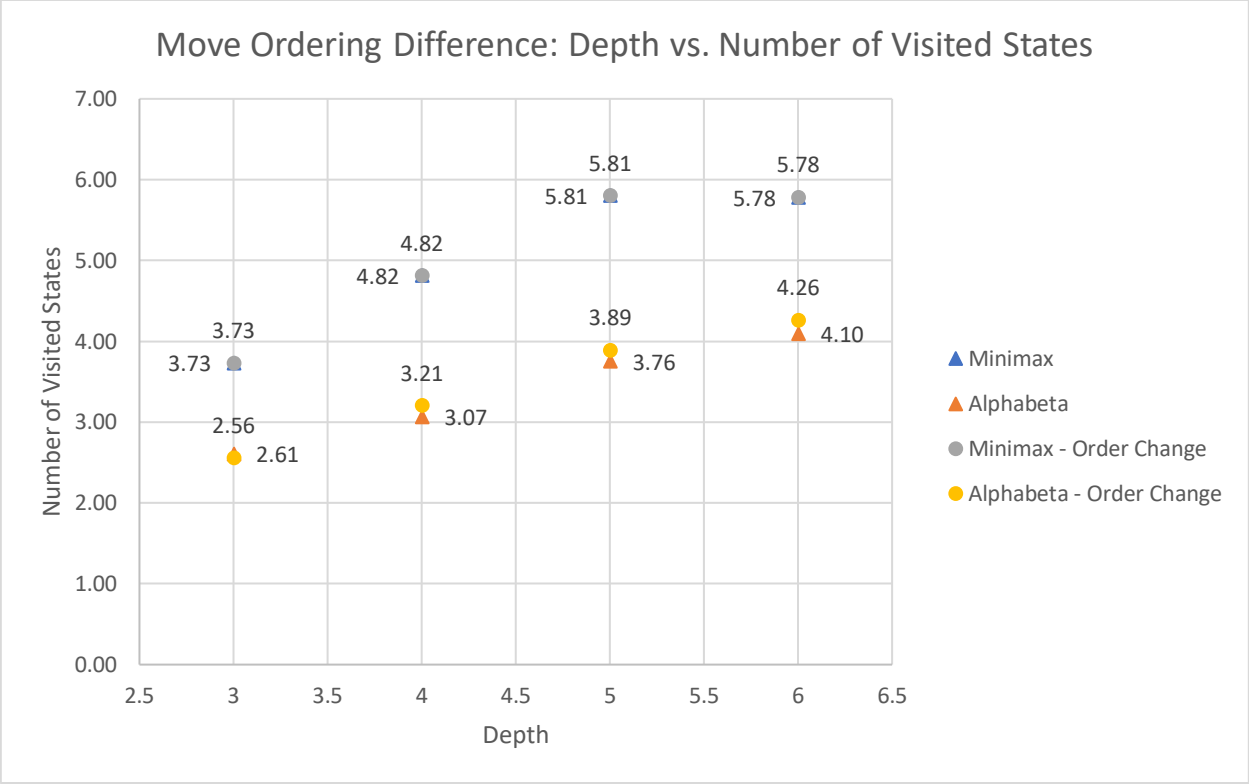

Figure 6 Move Ordering Difference: Depth vs. Number of Visited States

The results in Figure 6 support the aforementioned statementes regarding move ordering and the two adversarial search algorithms. The number of states visited during the minimax algorithm did not change, however, the number of states visited in the alphabeta algorithm did change.

In order to achieve optimal savings in the alphabeta algorithm the best possible moves would be explored first. However, one cannot know the best possible moves without computing them. Move ordering can be done through iterative deepening techniques in which the depth

is changed iteratively, and moves that were previously best are chosen first in the future. Another method is to compute the heuristic portion first, and explore moves with the highest heuristics first.

Question 3

If two agents are to play against each other for scenario (c) for a depth cutoff of 6, for which, if any, of minimax and/or alpha-beta pruning will the losing agent try to delay its defeat? Explain your answer.

If it is obvious that a game agent, say black, will win from the start, then a losing agent will try to delay its defeat in both the minimax and alphabeta pruning algorithms, however, it will likely be more efficient at delaying its defeat with the minimax algorithm.

The reason for this is that since minimax examines all possible moves, if the maximum value of any action available still results in a loss for white, the agent can choose the move that has the longest terminal path.

However, in the alphabeta algorithm, because branches that can't be better for the agent than other branches are pruned, there is an increased probability that a branch that results in a losing game, but also leads to a longer game, will be pruned. As such, the agent will likely use the minimax algorithm to always obtain the longest possible path to delay its defeat.

2 Part 2

Question 1

Provide a rationale for the choice of the improved evaluation function you used. In particular, explain how you implemented the behaviour described above.

The improved evaluation function includes a heuristic function as well as a linear multipler for the game state. Since the objective of the game is to create a 2x2 square of the player's pieces, the heuristic function that measures the desirability of the game state takes into account the proximity of player pieces to each other. There are two components to the heuristic function, a reward for having more pieces near a piece, and a penalty for having more of the opponent's pieces near a piece.

The reward component of the heuristic function adds to the value based on nearby pieces. If, for example, the desirability of the game state was being measured for player white, with pieces "O", then the heuristic would be calculated as follows. For each "O" on the board, a value of *2* is added to the heuristic function for every additional "O" in the 8 squares (if it's not on the edge) surrounding it. A value of *1* is added to the heuristic function for every additional "O" in the 16 squares (if it's not on the edge) surrounding each position. This is the reward component of the heuristic function. The logic behind this is that states in which there are as many of the player's pieces as close as possible to each other is more desirable as the probability that a winning state is fewer moves a way is high. It is clear why the numerical values for being closer are higher; the exact amounts were fine tuned through trial and error.

The penalty component of the heuristic function follows the same pattern of checking two layers around each piece, as above. However, instead of checking for "O" pieces for player white in the example above, it would check for "X" pieces. For every "X" piece, *0.25* would be subtracted from the overall heuristic value. The logic for this is that having opponent pieces nearby limits the number of moves, so the probability that a winning game state is nearby decreases. The exact amount of 0.25 was deduced through trial and error.

For the black player, the amounts and approach are the same as above, except they are negative.

If the agent is guaranteed a victory, it will take the shortest path possible, and if it is guaranteed a loss, it will take the longest path possible. This behaviour was outlined in the assignment instructions and was implemented through a separate function that keeps track of the values returned and terminal depths of all possible moves for the player. The status of the gamestate (+1 for white win, -1 for black win, 0 for neither) is multipled by constant of 50 before being added to the heuristic value and returned as the minimax/alphabeta value. The linear multiplier for the gamestate was chosen to be beyond the maximum range of the heuristic value, so those gamestates would always be prioritized. In the function to pick out the moves, the best move was persistently saved. However, if for example for white, the value returned by the algorithm was greater than 60, which would always be the case in a win, then the terminal depths would be compared and the move with the least terminal depth would be returned. The opposite would apply if the player was black. A very similar approach was taken to choose a delay move, except if for example for white, the value returned by the algorithm was less than 60, which would always be the case in a loss, then a separate move is persistently saved that has the longest terminal path.

Question 2

For a given search depth, does your improved evaluation function reduce the average number of nodes visited with 1) minimax? 2) alpha-beta? Illustrate appropriately.

In order to test if the heuristic value and improved evaluation function reduced the average number of nodes, the newly improved minimax and alphabeta algorithms will be tested with initial gamestate b) from Part 1. Minimax evaluates all possible positions up to a certain depth no matter what, so a change in the number of states is not expected there. Alphabeta pruning, however, prunes branches that cannot be the best move. If the evaluation function is indeed better, the best move should be arrived at faster, and a reduction in states visited is expected. The values will be displayed graphically below.

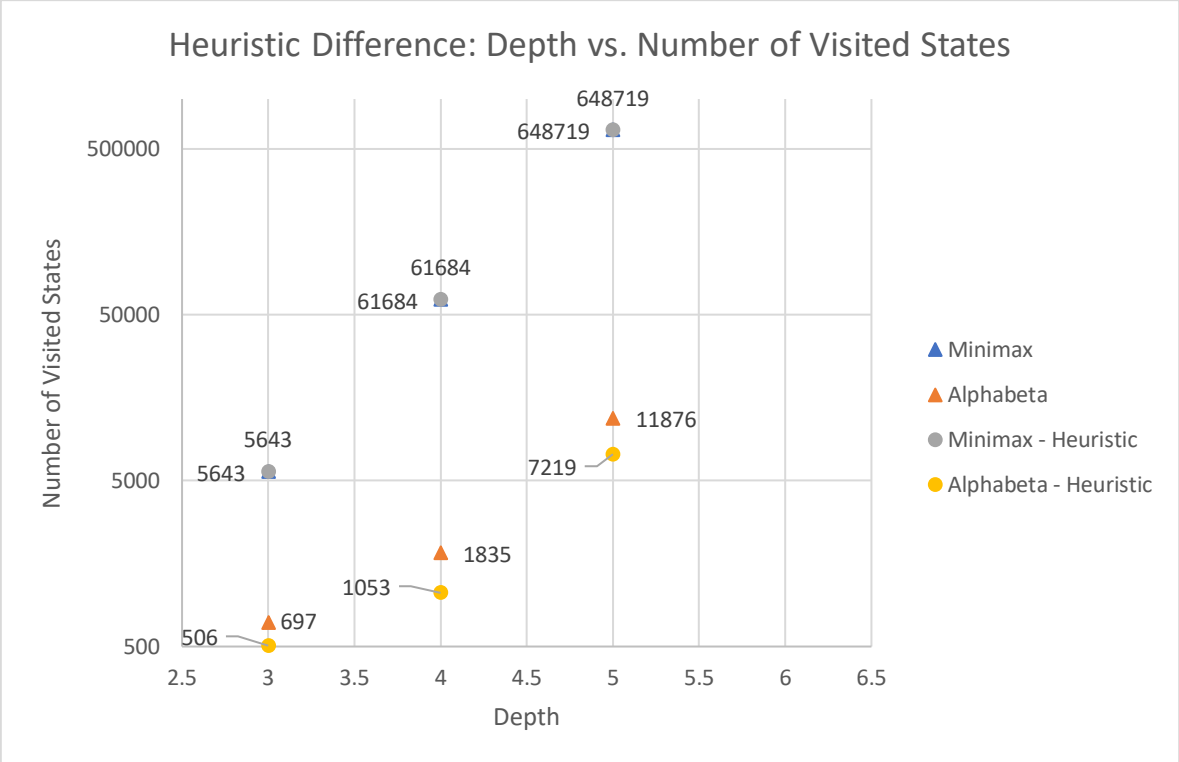

Figure 7 Heuristic Difference: Depth vs. Number of Visited States

The results above are in line with what would be expected given a heuristic that is an improvement.

The improved evaluation function is able to get to a better move faster because there is more detail and nuance in the desirability of a gamestate compared with the original evaluation of just -1, 0, and 1. In terms of time limitation, analyzing fewer game states and arriving at better moves faster allows the agent to play more optimally in a given period of time. In order to decrease the number of game states analyzed, a more complex and better performing heuristic can be tried and tested.

Question 3

Discuss the computational tradeoffs with the use of a more complex evaluation function with respect to the depth of the game tree that can be evaluated.

A more complex evaluation function increases the amount of time it takes for the agent to compute the best move out of a set of moves, however it is also more likely to choose a better move within that set.

On the other hand, a less complex evaluation function does not take as much time. As a result, it can explore a greater set of moves in the same amount of time as a more complex evaluation function. Of course doing this also requires more memory. These are the computational tradeoffs of memory and time when it comes to evaluation functions.

In terms of testing this, a more complex evaluation function was developed. The new evaluation function works in a similar way, except the number of layers are increased. Incrementing amounts of 1 are rewarded from the furthest layers to the piece being examined itself.

In order to analyze improved capability of a more complex function, the results will be normalized using the maximum attainable heuristic values. Overall, the evaluation function described in Question #1 has a maximum evaluation value of 100.

```
for position in player.positions:
    y = position[0]
    x = position[1]
    # Add to pieces if neighbouring piece is player's
    # Subtract from pieces if neighbouring piece is opponent's
    pieces = 0
    for i in range(-1, 2):
        for j in range(-1, 2):
            for k in range(1, layers):
                # Checking pieces around position
                if (0 > y + k*i or y + k*i > 6):
                    continue
                if (0 > x + k*j or x + k*j> 6):
                    continue

                surrounding_piece = self.board[y + k*i][x + k*j]
                if (surrounding_piece == sym):
                    pieces = pieces + (layers-k)*1
                elif (surrounding_piece != ' '):
                    pieces = pieces - (layers-k)*0.25
    heuristic += pieces
```

Figure 8 Dynamic heuristic function

Since the reward values for the heuristic are changing, the linear multiplier on the overall state of the game (win/loss) needs to change as well. The new, more complex heruistic will look one layer deeper, raising the maximum value of the heuristic to 90. As a result, the new linear multiplier for the game state will be 90. Overall, this evaluation function will have a maximum evaluation value fo 150.

Algorithm	Depth	Time (s)	Alphabeta Value	Normalized
Simple evaluation	3	0	16	0.16
Complex evaluation	3	1	30	0.17
Simple evaluation	4	5	32	0.32
Complex evaluation	4	8	62	0.34

Table 1: Analysis of evaluation function performance

Given a more complex evaluation function for the same depth, it seems as though a more complex evaluation function is able to perform better. However, the values are quite similar so testing at a large scale should be done to verify this.

The time tradeoff here using a more complex evaluation function may not be worth it in this case. It would depend on the use case, but for example if one had a time limit cutoff to make a move in 5 seconds, the simple evaluation function would result in significantly better performance than the complex one.

Question 4

For a depth cutoff of 4 for scenarios (a) and (c) in part 1, give a log of the game by two agents that use your developed heuristic. (In other words, run two instances of the code, with each playing one of the sides, and having its moves communicated via the game server, just as will be done in the class tournament).

The initial game states of (a) and (c) were used in conjunction with the provided game server to run a game between two agents using the heuristic portrayed above. The depth was given a cutoff of 4, and the heuristic used an evaluation function that went 2 layers

deep to check for the proximity of pieces. This was considered to be the “simple evaluation” in Question #3.

```
[[' ', 'O', ' ', ' ', 'X', 'X', 'X'],
 [' ', ' ', ' ', ' ', 'X', 'O', 'X'],
 [' ', ' ', ' ', ' ', ' ', ' ', ' '],
 ['O', ' ', ' ', ' ', ' ', ' ', ' '],
 [' ', ' ', ' ', ' ', ' ', ' ', ' '],
 [' ', ' ', ' ', ' ', ' ', ' ', 'O'],
 ['O', 'O', ' ', ' ', ' ', ' ', 'X']]
```

a)

```
[[' ', ' ', ' ', ' ', ' ', 'O', 'O'],
 [' ', ' ', ' ', ' ', 'O', 'X', 'X'],
 [' ', ' ', ' ', ' ', ' ', ' ', ' '],
 [' ', ' ', ' ', ' ', ' ', 'O', ' '],
 [' ', ' ', ' ', ' ', ' ', ' ', ' '],
 [' ', 'X', ' ', ' ', ' ', 'X', 'O'],
 [' ', 'X', 'X', ' ', ' ', ' ', 'O']]
```

c)

Figure 9 Initial game states for Question #4

Each agent maintains an internal belief state, and logs changes in that belief state and what move caused that change by listening and sending packets to the game server. The log is printed to the terminal, and those print statements were copied to present the logs below.

Of course following the rules of the game, white will start first. The log for the agents playing initially with a) is as follows.

```
1  [' ', 'O', ' ', ' ', 'X', 'X', 'X']
2  [' ', ' ', ' ', ' ', 'X', 'O', 'X']
3  [' ', ' ', ' ', ' ', ' ', ' ', ' ']
4  ['O', ' ', ' ', ' ', ' ', ' ', ' ']
5  [' ', ' ', ' ', ' ', ' ', ' ', ' ']
6  [' ', ' ', ' ', ' ', ' ', ' ', 'O']
7  ['O', 'O', ' ', ' ', ' ', ' ', 'X']
8  Game started!
9  White's move: 21E2
10
11 [' ', ' ', ' ', 'O', 'X', 'X', 'X']
12 [' ', ' ', ' ', ' ', 'X', 'O', 'X']
13 [' ', ' ', ' ', ' ', ' ', ' ', ' ']
14 ['O', ' ', ' ', ' ', ' ', ' ', ' ']
15 [' ', ' ', ' ', ' ', ' ', ' ', ' ']
16 [' ', ' ', ' ', ' ', ' ', ' ', 'O']
17 ['O', 'O', ' ', ' ', ' ', ' ', 'X']
18 Black's move: 77W2
19
20 [' ', ' ', ' ', 'O', 'X', 'X', 'X']
21 [' ', ' ', ' ', ' ', 'X', 'O', 'X']
22 [' ', ' ', ' ', ' ', ' ', ' ', ' ']
23 ['O', ' ', ' ', ' ', ' ', ' ', ' ']
24 [' ', ' ', ' ', ' ', ' ', ' ', ' ']
25 [' ', ' ', ' ', ' ', ' ', ' ', 'O']
26 ['O', 'O', ' ', ' ', 'X', ' ', ' ']
27 White's move: 76W2
28
29 [' ', ' ', ' ', 'O', 'X', 'X', 'X']
30 [' ', ' ', ' ', ' ', 'X', 'O', 'X']
31 [' ', ' ', ' ', ' ', ' ', ' ', ' ']
32 ['O', ' ', ' ', ' ', ' ', ' ', ' ']
33 [' ', ' ', ' ', ' ', ' ', ' ', ' ']
34 [' ', ' ', ' ', ' ', 'O', ' ', ' ']
35 ['O', 'O', ' ', ' ', 'X', ' ', ' ']
```

```
36 Black's move: 57W2
38 [' ', ' ', ' ', 'O', 'X', 'X', 'X']
39 [' ', ' ', ' ', ' ', 'X', 'O', 'X']
40 [' ', ' ', ' ', ' ', ' ', ' ', ' ']
41 ['O', ' ', ' ', ' ', ' ', ' ', ' ']
42 [' ', ' ', ' ', ' ', ' ', ' ', ' ']
43 [' ', ' ', ' ', ' ', 'O', ' ', ' ']
44 ['O', 'O', 'X', ' ', ' ', ' ', ' ']
45 White's move: 56W3
46
47 [' ', ' ', ' ', 'O', 'X', 'X', 'X']
48 [' ', ' ', ' ', ' ', 'X', 'O', 'X']
49 [' ', ' ', ' ', ' ', ' ', ' ', ' ']
50 ['O', ' ', ' ', ' ', ' ', ' ', ' ']
51 [' ', ' ', ' ', ' ', ' ', ' ', ' ']
52 [' ', 'O', ' ', ' ', ' ', ' ', ' ']
53 ['O', 'O', 'X', ' ', ' ', ' ', ' ']
54 Black's move: 72S1
55
56 [' ', ' ', ' ', 'O', 'X', 'X', 'X']
57 [' ', ' ', ' ', ' ', 'X', 'O', ' ']
58 [' ', ' ', ' ', ' ', ' ', ' ', 'X']
59 ['O', ' ', ' ', ' ', ' ', ' ', ' ']
60 [' ', ' ', ' ', ' ', ' ', ' ', ' ']
61 [' ', 'O', ' ', ' ', ' ', ' ', ' ']
62 ['O', 'O', 'X', ' ', ' ', ' ', ' ']
63 White's move: 14S2
64
65 [' ', ' ', ' ', 'O', 'X', 'X', 'X']
66 [' ', ' ', ' ', ' ', 'X', 'O', ' ']
67 [' ', ' ', ' ', ' ', ' ', ' ', 'X']
68 [' ', ' ', ' ', ' ', ' ', ' ', ' ']
69 [' ', ' ', ' ', ' ', ' ', ' ', ' ']
70 ['O', 'O', ' ', ' ', ' ', ' ', ' ']
71 ['O', 'O', 'X', ' ', ' ', ' ', ' ']
```

Figure 10 Agents using heuristic with alphabeta against each other for a)

The game above ends in 7 moves, with white taking the win in the bottom left corner. In order to test if the heuristic actually made a difference, the game was also run using alphabeta without any heuristic. Without any heuristic, the game never actually terminated and was caught in an infinite loop of the players making the same moves in a cycle. It is clear that the developed heuristic and improved evaluation function aids the agent.

The log for the agents playing initially with c) is as follows.

```
1  [' ', ' ', ' ', ' ', ' ', 'O', 'O']
2  [' ', ' ', ' ', ' ', 'O', 'X', 'X']
3  [' ', ' ', ' ', ' ', ' ', ' ', ' ']
4  [' ', ' ', ' ', ' ', ' ', 'O', ' ']
5  [' ', ' ', ' ', ' ', ' ', ' ', ' ']
6  [' ', 'X', ' ', ' ', ' ', 'X', 'O']
7  [' ', 'X', 'X', ' ', ' ', ' ', 'O']
8  Game started!
9  White's move: 64N1
10
11 [' ', ' ', ' ', ' ', ' ', 'O', 'O']
12 [' ', ' ', ' ', ' ', 'O', 'X', 'X']
13 [' ', ' ', ' ', ' ', ' ', 'O', ' ']
14 [' ', ' ', ' ', ' ', ' ', ' ', ' ']
15 [' ', ' ', ' ', ' ', ' ', ' ', ' ']
16 [' ', 'X', ' ', ' ', ' ', 'X', 'O']
17 [' ', 'X', 'X', ' ', ' ', ' ', 'O']
18 Black's move: 66W1
19
20 [' ', ' ', ' ', ' ', ' ', 'O', 'O']
21 [' ', ' ', ' ', ' ', 'O', 'X', 'X']
22 [' ', ' ', ' ', ' ', ' ', 'O', ' ']
23 [' ', ' ', ' ', ' ', ' ', ' ', ' ']
24 [' ', ' ', ' ', ' ', ' ', ' ', ' ']
25 [' ', 'X', ' ', ' ', 'X', ' ', 'O']
26 [' ', 'X', 'X', ' ', ' ', ' ', 'O']
27 White's move: 77W3
28
29 [' ', ' ', ' ', ' ', ' ', 'O', 'O']
30 [' ', ' ', ' ', ' ', 'O', 'X', 'X']
31 [' ', ' ', ' ', ' ', ' ', 'O', ' ']
32 [' ', ' ', ' ', ' ', ' ', ' ', ' ']
33 [' ', ' ', ' ', ' ', ' ', ' ', ' ']
34 [' ', 'X', ' ', ' ', 'X', ' ', 'O']
35 [' ', 'X', 'X', 'O', ' ', ' ', ' ']
36 Black's move: 56W2
37
38 [' ', ' ', ' ', ' ', ' ', 'O', 'O']
39 [' ', ' ', ' ', ' ', 'O', 'X', 'X']
40 [' ', ' ', ' ', ' ', ' ', 'O', ' ']
41 [' ', ' ', ' ', ' ', ' ', ' ', ' ']
42 [' ', ' ', ' ', ' ', ' ', ' ', ' ']
43 [' ', 'X', 'X', ' ', ' ', ' ', 'O']
44 [' ', 'X', 'X', 'O', ' ', ' ', ' ']
```

Figure 11 Agents using heuristic with alphabeta against each other for a)

The game above ends in 5 moves, with black taking the victory in the bottom-left corner. Much like above in order to test if the heuristic actually made a difference, the game was also run using alphabeta without any heuristic. Without any heuristic, the game ended in 11 moves, with black taking the victory in the same way. It is clear that the developed heuristic and improved evaluation function aids the agent as for the same depth, the agent was able to play better and win the game in less moves.

3 References

[1] Russell, Stuart J.; Norvig, Peter (2003), Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, ISBN 0-13-790395-2