Yujin Li
260968957

Part 1

1. State a

Depth cutoffs	Total numbers of states visited using Minimax	Total numbers of states visited using Alpha-Beta Pruning
3	7379	756
4	113635	1591
5	3048195	25559
6	52217983	51275

State b

Depth cutoffs	Total numbers of states visited using Minimax	Total numbers of states visited using Alpha-Beta Pruning
3	5644	669
4	98761	1411
5	2129587	33549
6	42169252	65860

State c

Depth cutoffs	Total numbers of states visited using Minimax	Total numbers of states visited using Alpha-Beta Pruning
3	5421	363
4	109672	5136
5	2063480	14987
6	45639613	262449

2. In Minimax implementation, the number of states visited remains the same regardless of the order of evaluation. Since base on the algorithm, Minimax would evaluate all states generated. In Alpha-Beta Pruning algorithm, the number of states visited depends on the order of new states generated. The order of states generated would affect the branches being “pruned” using the algorithm.

   In question 1, the order of new states generated is N1->S1->W1->E1->N2… (if the move is valid), here I switch it to N3->S3->W3->E3->N2… use State a as an example

   State a using evaluation order N3->S3->W3->E3->N2…

Depth cutoffs	Total numbers of states visited using Minimax	Total numbers of states visited using Alpha-Beta Pruning
3	7379	716
4	113635	1514
5	3048195	20982
6	52217983	42658

Yujin Li
260968957

By running test on State a, we can observe that the number of states visited remains the same while using Minimax algorithm but slightly changed using Alpha-Beta Pruning.

3.

Assuming that we have the simple heuristic, that is 1 for win, -1 for lost and 0 for neutral, taking State a as an example where white (O) is 100% wining and black(X) is 100% losing

Depth cutoffs	Number of moves using Minimax	Number of moves using Alpha-Beta Pruning
6	O: 5 moves; X: 4 moves	O:5 moves; X:4 moves

Losing Agent is not trying to delay its defeat, it takes the first command from the possible moves that I assigned to it. If two agents are to play against each other, neither agent will actively try to delay its defeat if no further heuristic function is implemented.

The Minimax algorithm explores all possible moves up to the specified depth limit and assume both players will make the best moves to maximize their chances to win. It will just return the first move with the highest score, number of steps taken (i.e. depth of the tree) is not a factor that would impact the decision. When a losing state is reached, since all losing states gives the same score, which is -1, therefore, it would just consider the first move that we assign to it as the best move (best move would only be updated when the score is higher than the previous one).

Same thing for Alpha-Beta Pruning, Alpha-Beta Pruning prunes some branches that are unnecessary to be evaluated but number of steps taken (i.e. depth of the tree) is still not a factor that would be consider, as the result, it would behave the same way as the Minimax algorithm, except that Alpha-Beta Pruning provides us answer in a shorter amount of time.

Therefore, both Minimax and Alpha-Beta Pruning would not try to delay its defeat without further heuristic implemented.

Part 2

1. Here is the heuristic function that I implemented: I evaluate the value of each chess and sum them up to represent the score of each board, if a winning condition is reached, 1000, or -1000 would be returned, depends on who wins (1000 for winning and -1000 for losing).

   For order of generation:

   Instead of generating moves from smaller number to larger number, now it is the reversed order.

   For each chess:

   a. if an 'L' shape is formed, in addition to the 1 point added in a, another 10 points would be added

Yujin Li
260968957

b. if an 'L' shape is formed but the 4th slot (that is needed to form a 2x2 square) is taken by the opponent ('O' in this case), 10 points would be removed (in any of the case in Figure 2.1)

X	O		O	X		X	X		X	X	
X	X		X	X		X	O		O	X	

Figure 2.1

c. if an 'L' shape is formed, the 4th slot is empty (e.g., the red part one in the Figure 2.2); if an ally chess ('X' in this case), is located in anyone of the yellow slots and no enemy chess ('O' in this case) is blocking its way to red slot, 20 points would be added (this rule also applies on the rotated version of Figure 2.2)

X				
X	X			

Figure 2.2

2. State a

Depth cutoffs	Minimax	Minimax with heuristic	Alpha-Beta Pruning	Alpha-Beta Pruning with heuristic
3	7379	7379	756	866
4	113635	113635	1591	2357
5	3048195	3048195	25559	22209
6	52217983	52217983	51275	48354

State b

Depth cutoffs	Minimax	Minimax with heuristic	Alpha-Beta Pruning	Alpha-Beta Pruning with heuristic
3	5644	5644	669	956
4	98761	98761	1411	3739
5	2129587	2129587	33549	32034
6	42169252	42169252	65860	89023

State c

Depth cutoffs	Minimax	Minimax with heuristic	Alpha-Beta Pruning	Alpha-Beta Pruning with heuristic
3	5421	5421	363	513
4	109672	109672	5136	8361

Yujin Li
260968957

5	2063480	2063480	14987	29596
6	45639613	45639613	262449	387677

From the observation, we can see that:

1) There isn't any change after applying heuristic to Minimax algorithm.
2) In most of the cases, Alpha-Beta Pruning with improved evaluation function has larger number of nodes visited at each given depth compare to Alpha-Beta Pruning with simple heuristic implemented in part 1, but still has much smaller number of nodes visited compare to Minimax Algorithm.

Heuristic function (evaluation function) does not have any impact on Minimax algorithm due to the fact that the Minimax algorithm will evaluate all nodes on the game tree no matter what happened, therefore it is not a good idea to put heuristic on Minimax algorithm because it would just increase the computational cost.

For Alpha-Beta Pruning, the heuristic function does have a huge impact. Generally, a more accurate heuristic function can indirectly help alpha-beta pruning by providing better estimates of game state values, potentially leading to more effective pruning and resulting in a smaller number of nodes visited. However, in our case, the simple heuristic implemented in Part 1 gives too many states with the same score (neutral state, resulting in 0), this causes alpha-beta pruning to prune many branches, including those that might lead to a win. In the improved evaluation function, we provided a better estimation of the game state, although more nodes need to be evaluated at each given depth but it would take less steps to win.

3. From the observation that we have in Part2.2, there is a trade-off between the complexity of the evaluation function and the depth of the game tree that can be evaluated. A more complex and more competent heuristic can provide more accurate estimation when the wining condition (or losing condition) cannot be reached within the given depth. However, in terms of efficiency, a complex heuristic may limit the depth of the search due to increase computation time (time increased by having more nodes to evaluate and the time for running evaluation function itself). Therefore, when the computation time is limited, we need to decide whether to have a simpler but faster algorithm to make a "deeper exploration", or have a more accurate heuristic but slower algorithm to have a "shallower exploration". The balance between accuracy and efficiency is crucial. In addition, if we are using a simple but faster evaluation function, memory may be an issue because making deeper search requires more memory.

4. For a depth cutoff of 4, Figure 2.3 indicates the move of 2 AI agents in State a; Figure 2.4, Figure 2.5, Figure 2.6, Figure 2.7, Figure 2.8 indicates the move in State c.

   For a depth cutoff of 5, Figure 2.9 indicates the move of 2 in State a; Figure 2.10 indicates the move in State b

Yujin Li
260968957

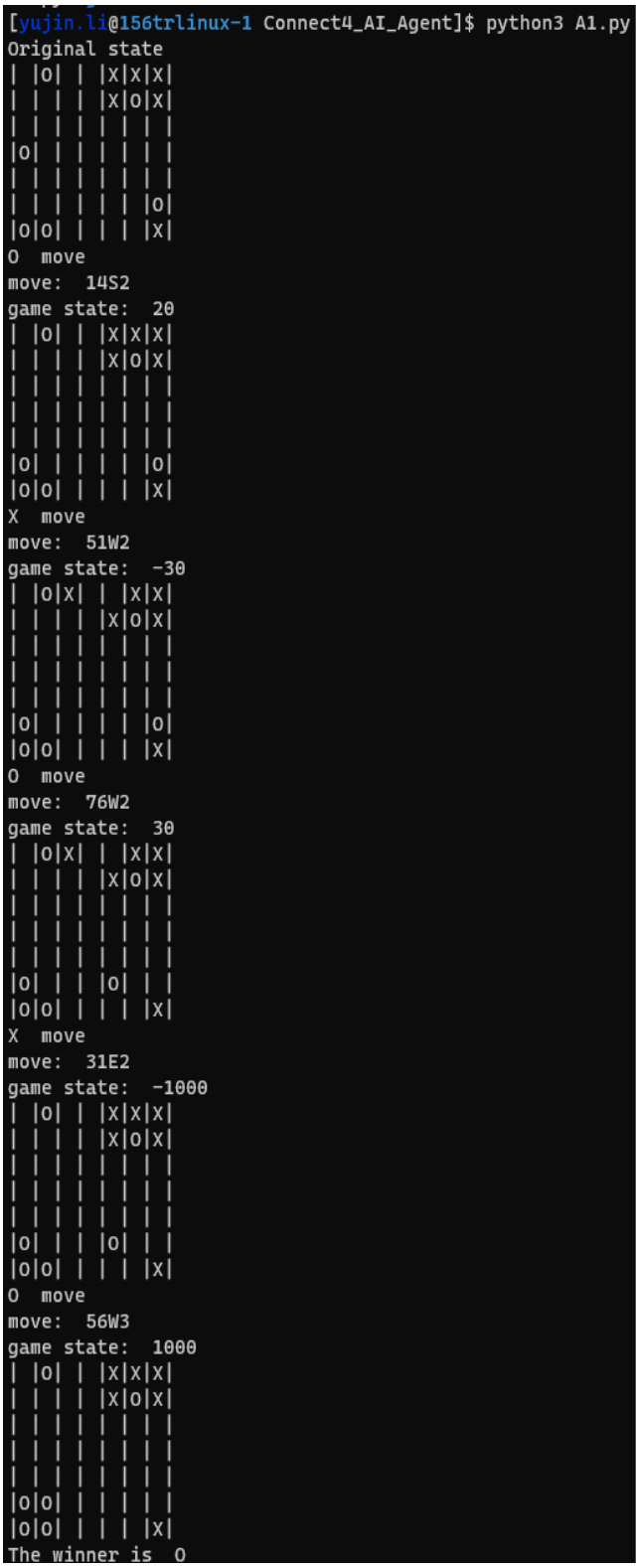

Figure 2.3

Yujin Li
260968957

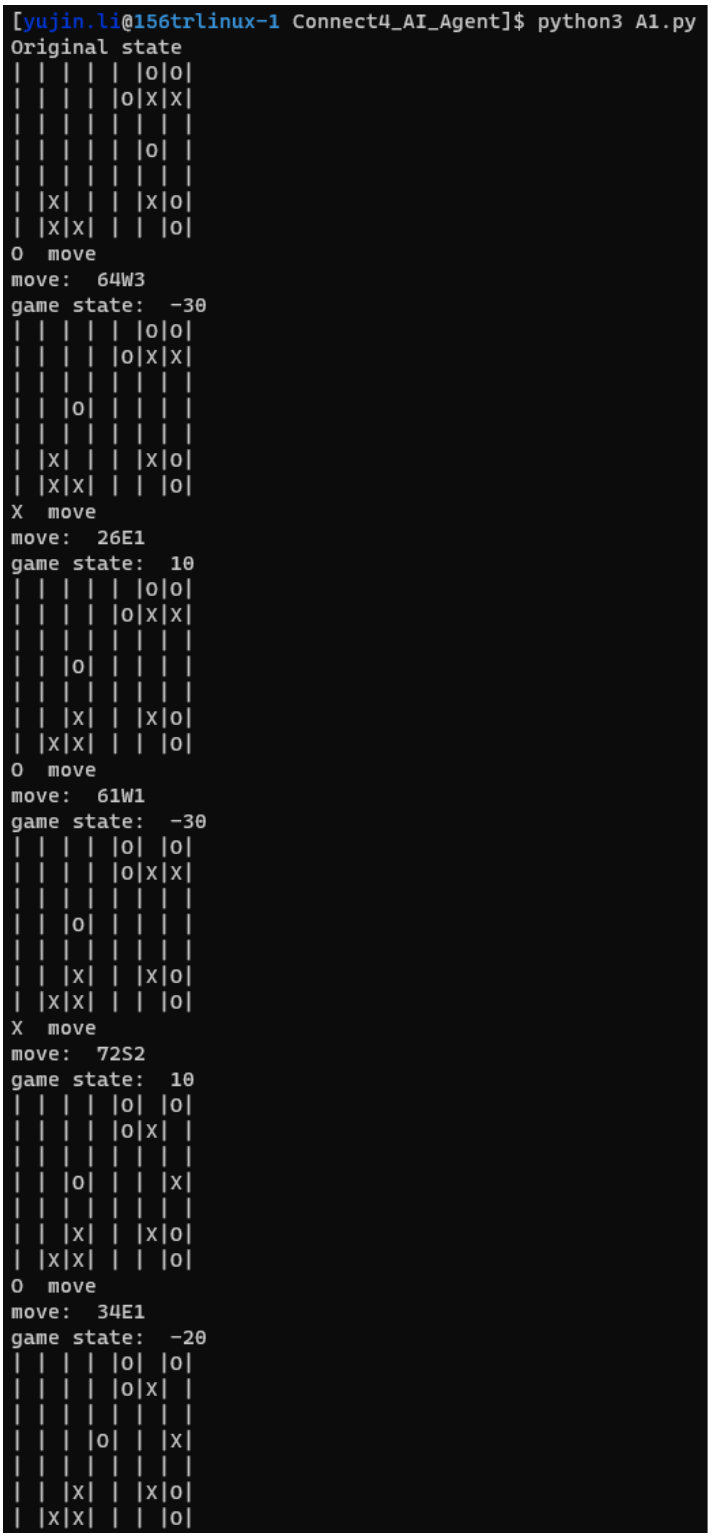

Figure 2.4

Figure 2.5

Yujin Li
260968957

Figure 2.6

Figure 2.7

Figure 2.8

Yujin Li
260968957

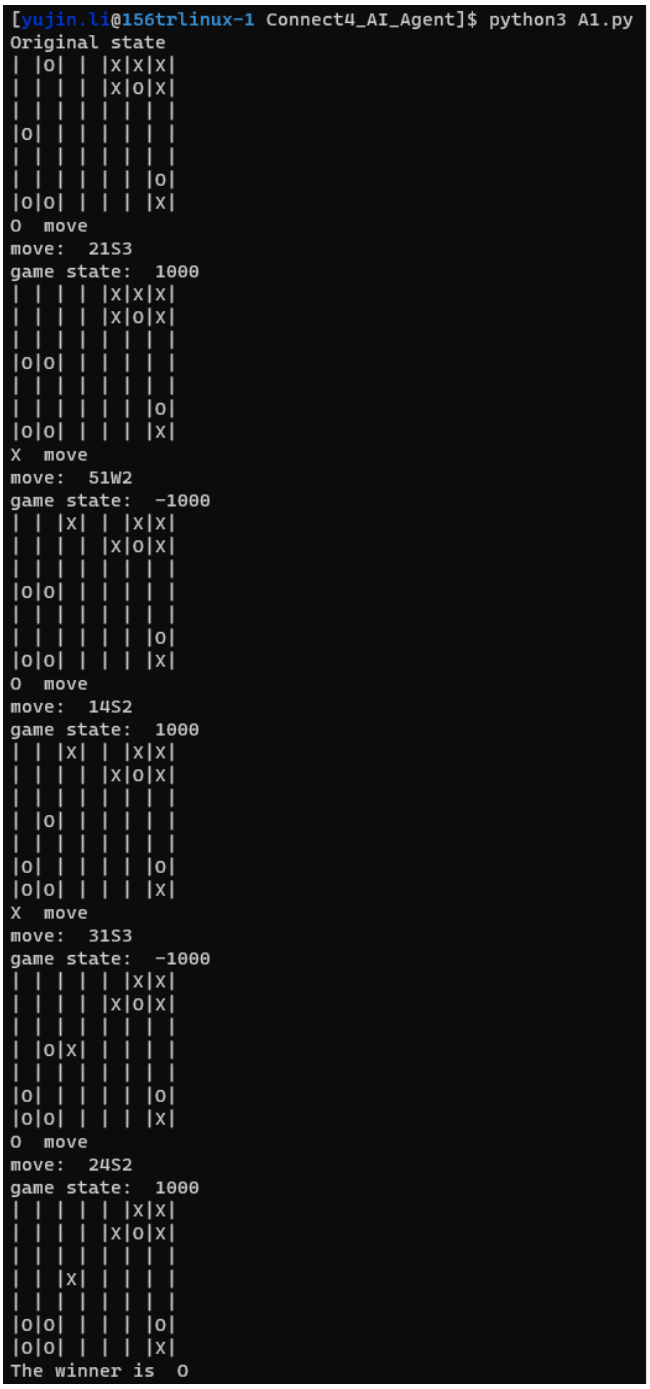

Figure 2.9

Yujin Li
260968957

```
[yujin.li@156trlinux-1 Connect4_AI_Agent]$ python3 A1.py
Original state
| | | | | |o| |
| | | | | |x| |
| | | | |x|x| |
| | | | | |o| |
| | | | | |x| |
| |o| | | |o| |
|x|x|o|o| | | |
O  move
move:  66W2
game state:  1000
| | | | | |o| |
| | | | | |x| |
| | | | |x|x| |
| | | | | |o| |
| | | | | |x| |
| |o| |o| | | |
|x|x|o|o| | | |
X  move
move:  62W1
game state:  -1000
| | | | | |o| |
| | | | |x| | |
| | | | |x|x| |
| | | | | |o| |
| | | | | |x| |
| |o| |o| | | |
|x|x|o|o| | | |
O  move
move:  26E1
game state:  1000
| | | | | |o| |
| | | | |x| | |
| | | | |x|x| |
| | | | | |o| |
| | | | | |x| |
| | |o|o| | | |
|x|x|o|o| | | |
The winner is  O
```

Figure 2.10