Assignement 1

In the following, we answer the raised questions. The codes as well as a readme file and a sample game log are attached in the same rar file.

Part I

1. We run Mont Carlo simulations (N=20) for different values of the cut-off depth d to evaluate the average time required by the minimax algorithm, minimax(self, player,depth), to perform the state space expansion and evaluation of the game tree for different d's. The average times are grouped in the following table and plotted in figure 1:

Cut off depth	2	3	4	5	6
Time (s)	0,082085	0,606531	4,481689	33,11545	148,4132

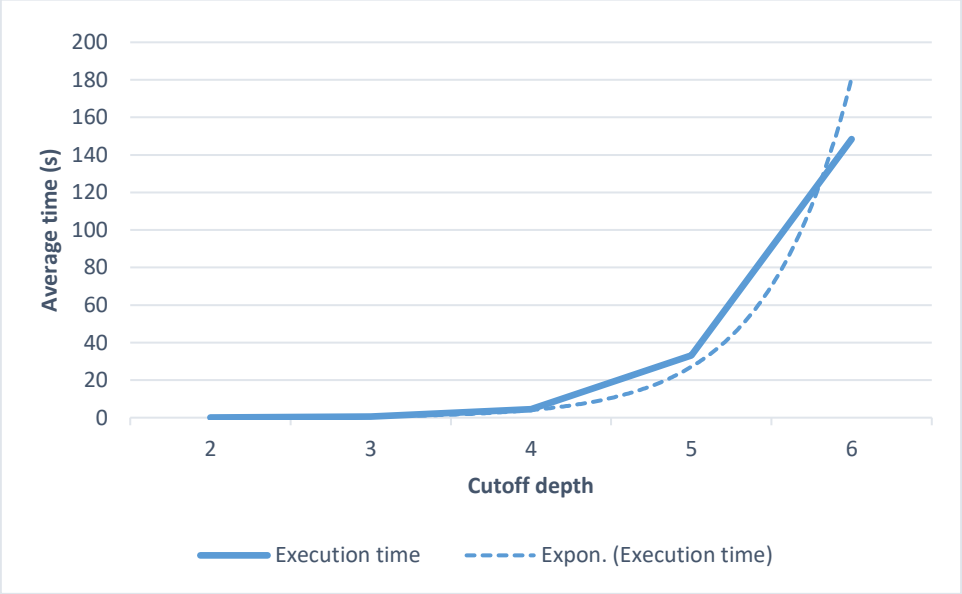

Figure 1 Average execution time for the minimax algorithm for different cut-off depth

This plot shows the exponential (time) cost of going in depth.

2. We implement now the alpha-beta pruning, the the method__minimaxAlphaBeta(self, player,depth,alpha,beta), and redo as previously done to visualize the performance changes. Results in figure 2 show an important improvement which allows to go up to depth=6 while staying under the limit of 10s.

Cut off depth	3	4	5	6	7
Time (s)	0,096981	0,188147	0,801022	5,124467	17,74057

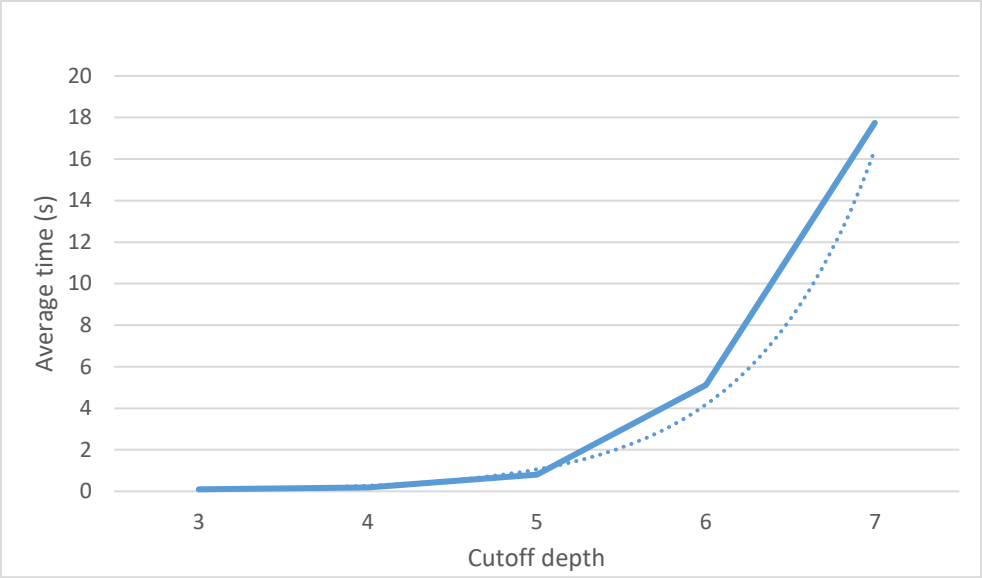

Figure 2 Average execution time for the minimax algorithm with alpha beta pruning for different cut-off depth

Part II

3. Going more in depth allows the agent to predict more future moves and then has better knowledge and higher chances but this comes at a high cost of an exponential increase in computational time. However, since the constraint in our game is the time limit allowed, we can not go much in depth as we have seen in Part 1. Thus, here comes the importance of heuristics that provide evaluation function for non-terminal nodes. The more the heuristic could give a good prediction, the less depth we need to go. But in the other hand, better evaluation means more complexity and thus more execution time while it still 'estimation' (compared to the 'certainty' while going in depth to terminal nodes), hence the trade-off between the two, depth and complexity of heuristics. The idea is to go as much deep as time allows when using a 'good enough' heuristic (through simulations). In our case, with the initial heuristic, depth = 6 allows us to stay under the 10 seconds limit (while d= 7 does not) for which we develop a more complex heuristic. The idea is to consider several criteria like:

   - Number of blocked opponent pieces: Difference between the number of opponent's and yours blocked pieces (pieces which don't have an empty adjacent point)
   - Number of runs: Difference between the number of yours and opponent's 2 piece configurations
   - Number of possible moves: Difference between the number of yours and opponent's possible moves
   - …

   And combine them with weighting, i.e., the evaluation function would be a linear combination of these criteria. The weighting factor would be determined through simulation completion for different configurations to find the best.

Part III

4. To be able to run the 7x6 grid with the initial configuration proposed, we add the following code lines in the beginning to ask the user for the grid size:

```
board_size = raw_input("Please enter 1 for 5*4 board or 2 for 7*6 board:")
        if board_size=='1':
                nbr_cols=5
                nbr_rows=4
        else:
                nbr_cols =7
                nbr_rows =6
g = Game(nbr_cols, nbr_rows)
```

So that the first the start would be:

```
Please enter 1 for 5*4 board or 2 for 7*6 board:1
Enter gameID and player's color: game789 white
This is the current board:
0 . . . 1
1 . . . 0
0 . . . 1
1 . . . 0
()
white's turn:
```

```
Please enter 1 for 5*4 board or 2 for 7*6 board:2
Enter gameID and player's color: game171 white
This is the current board:
. . . . . .
. 0 . . . 1 .
. 1 . . . 0 .
. 0 . . . 1 .
. 1 . . . 0 .
. . . . . .
()
white's turn:
```