Project 1 Report

Yanhao Ding

September 30, 2018

Contents

1 Basic Tic-Tac-Toe **1**
- 1.1 Class Board . . . 1
- 1.2 Structure of Program . . . 2
- 1.3 Detailed utility analysis . . . 3
- 1.4 Program Performance Analysis : Will AI Never Lose? . . . 3

2 Advanced Tic-Tac-Toe **4**
- 2.1 Class Board . . . 4
- 2.2 Class NBoard . . . 4
- 2.3 Structure of the program . . . 5
  - 2.3.1 Outline of the program . . . 5
  - 2.3.2 Heuristic function implementation . . . 6
  - 2.3.3 Heuristic function performance . . . 7
  - 2.3.4 Alpha-beta pruning . . . 8
  - 2.3.5 Program terminate test . . . 8
- 2.4 Program Performance Analysis : Will AI Lose? . . . 9

3 Super Tic-Tac-Toe **9**
- 3.1 Updating terminate test . . . 9
- 3.2 Program Performance Analysis . . . 10

The project is developed using Java. The attached README file provides the instructions for running the project. This report outlines the procedure to develop the project and decribes the performance of the program.

1 Basic Tic-Tac-Toe

1.1 Class Board

Class **Board** was developed to represent the state of the TTT board arrangement of X's and O's. The **Board** coded in a separated file. The constructor of the **Board** class initialize a two dimensional String array with symbol * (star), which represent the grid is blank now. Another constructor make a deep copy of the input **Board**, which allows the defensive copy in other classes. The **Board** class includes several public method : getElement(), setElement(), PrintBoard(), PrintBoardLine(), getDimension(), setDead(), Heuristic(), and private method : getScore(); I will elaborate the usage of these method in the following paragraph. The **Board** class does not contains the information of whose turn to play, this part is done in the constructor of class **TTTGame**.

1.2 Structure of Program

The main part of the game is developed in the class **TTTGame** which also include the main method in this class. The fields of the class includes : board(Board type), step(int), sign(int). The board represents the states of arragement of X and O. step records how many steps we have play. sign is a important variable to control the postive or negative value of utility, the value is either 1 or -1. For example, we can set utility of X win as 1*sign and O win as -1*sign. If AI plays X, sign = 1. The utility of AI win gives 1. If AI plays O, sign = -1. The unitity of AI win also give 1. Using this method, the MINIMAX algorithm will work properly no mater which character AI play.

At the beginning of the game, the program will ask you which role you want play "X" or "O". Response of "x" and "o" is also acceptable. But other inputs are not valid. If you input the invalid string, the program will politely ask you to rethink to play "X" or "O". If your input is "X", the program will ask you the first step you want to play. The inteface in terminal is shown as Fig 1.

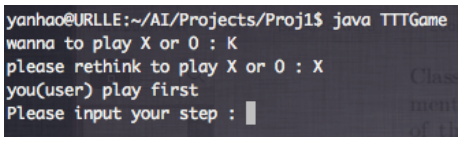

Figure 1: Terminal interface to choose X and invalid

If you choose play to play "O", the AI will play "X". After AI think which step will take, the program will print the board and ask your next step, shown in Fig 2.

Figure 2: Terminal interface to choose O

If your choose to play "O". AI need to think and make the first move. Otherwise, AI need to think and make the step after your first step. The thinking process is the implementation of MINIMAX algorithm. The final utility is given by the the method **TerminalTest()**. I did not find better method to test if the anyone has won the game. I just count everyline, column and diagonal to see whether a line have been formed.

The possible actions that AI can make at certain state of board are stored in an array. Each element of the action array is one possible state. Every element go deeper and repeat the process recursively unitl the terminal test finds the game is terminated and return the utility value. At each level, the program will chooose MIN or MAX value of utility and return to the above level. Once the utility value reachs the first level, the program will choose the best move and update the board with the appropriate mark at chosen position. The program will print updated board to the terminal and reminder next player to chooose position. After human player chose one position, the program will also print the updated board. Then AI think, AI choose step, update. The system will check if the human input step is valid or not. If invalid, the program will ask you input again. The one example loop of this process is shown as Fig 3.

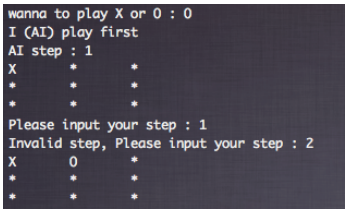

Figure 3: One step play between Human and AI

Above description shows the basic component and function of the program. Next, let us test the performance of the program.

1.3 Detailed utility analysis

I will analyze several cases to illustrate the AI make good decision by print out the utility values. The examples are shown is Fig 4.

Let us examize left panel first. When we analyze the situation, we assumes that the opponent will take the best move. Aftet human take step 1, AI will "think" which step to take, the utility of the grid left (8 spaces) is all -1 except at grid 5(the fouth) is 0, which means AI will lose to take step except 5. If AI take step 5, the best results is tie. This performance is in good agreement with my understanding. Under this situation, the only viable step is 5. All other steps will result in the loss of the game. After AI take action, I take step 2, which leave AI the only feasible solution is 3(the first utility). We can clearly see from utility value that AI make correct decision to take step 3.

Moreover, the right panel of figure shows that AI take first step at 1 and I take step at 2. From the utiltiy output, we can tell that AI think it is good to take step at 4 or 5. This is correct move. Then AI take one of the two steps 4. After that, I take step at 7 to prevent Ai from winning. AI think it will win by step at 5. This is also a correct desicion. It is also very interesting to analyze other moves, if AI take step 3(the first utility), if I take step at 8. AI lose. If AI take step 3, 6 ,9. same situation if I take 8 next. AI also thinks if it take step at 8. The results is tie. This is also correct for that we can see that I need step 8 to win.

From above four examples, we can see that the AI can make correct decisions. I think the implementation of MINIMAX algorithm is correct the performance is acceptable.

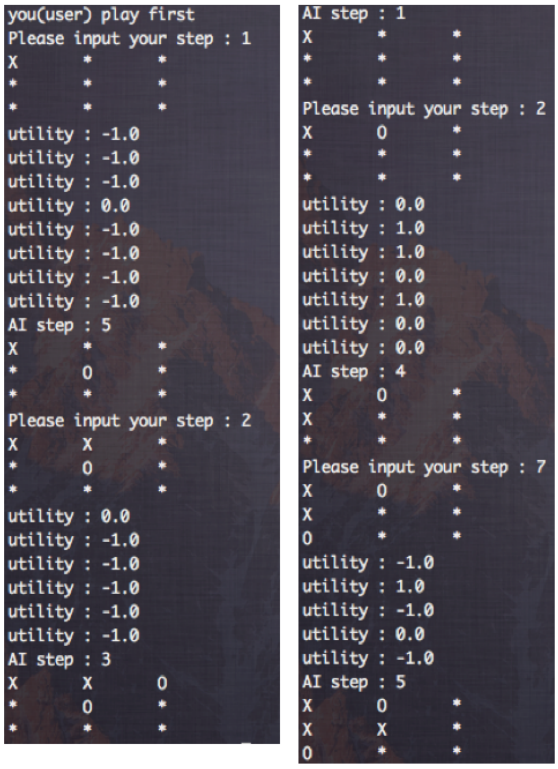

Figure 4: Detailed Utility Analysis

1.4 Program Performance Analysis : Will AI Never Lose?

Since we do not use any heuristic evaluation function, the AI thinks every possible step and make the decision. So it is impossible to beat AI. The best outcome is tie. In simple TTT game, the "thinking" time for AI is less one 1 second. First, let AI play first. After AI take position 1, we

could choose position 5 or other. I found that If we do not choose 5, I will lose. If I choose 5, the results will be tie. The two situation is shown as Fig 5. Under this circumstance, AI will not lose.

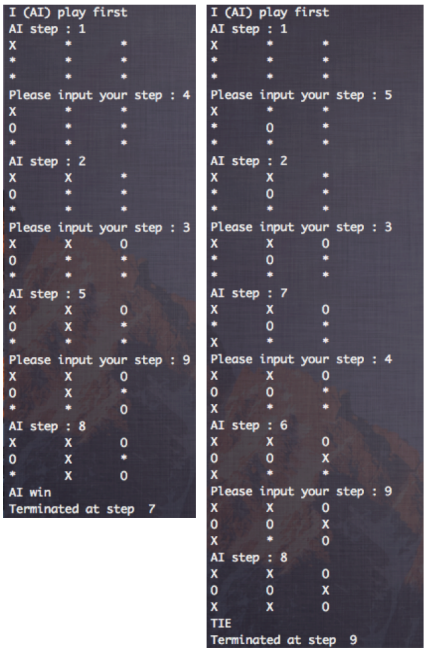

Figure 5: AI play X (X play first)

The second situation is human play first. The best outcome for human is tie. AI will not lose either. Two example plays of this type is shown as Fig 6. In the left panel of the figure, I take step 5 first, then AI take 1, I take step 2, AI realize the danger and take step 8 to avoid loss. I take step 4, AI take step 6 to avoid another potential loss. After this step, the game will end up with tie. In the right panel of the figure, I take step 1 first, AI will take step 5 because if not AI will lose. I took step 2, AI take step 3. I take step 7, AI take step 4, I take step 6, AI take step 8. The results is tie too. After test several case, I find that the AI is invincible in basic TTT game.

2 Advanced Tic-Tac-Toe

Main structure of code from Part1 can be used in this part of project. However, we need to add Nboard class to represent the state of play, implement advanced techniques like alpha-beta pruning, depth-limited-search and provides the heuristic function.

2.1 Class Board

The class **Board** is exactly same as the board from part 1. In this part, the Board acts as the sub-board of the large board, which represented by **Nboard class**.

2.2 Class NBoard

Class **Nboard** represents the arrangement of class **Board** on 3*3 grids. The Nboard class contains three fields: an 2D array of **Board** type, an 2D array of String type to track whether one sub-board has been won by any user. The constructor of this class will initiate a 2D array, and each element of the array is one object of class **Board**. The element of each Board is start with symbol “*””. Another overloaded constructor make a deep copy of an object of **Nboard**. The class also includes several methods :

- getBoard() : get one element of Nboard. The returned element is one object of Board.
- setBoard() : set Board element when the whole board is win by user, only useful when in part3.
- PrintNBoard () : print out the whole 9 Board to terminal

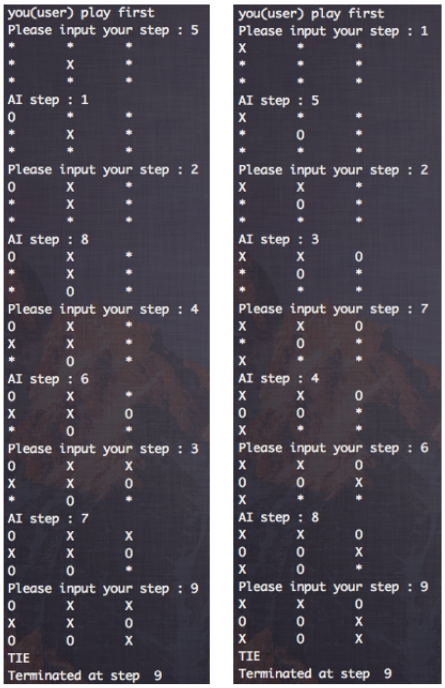

Figure 6: HUMAN play X (X play first)

One example of the 9board is shown as Fig 7.

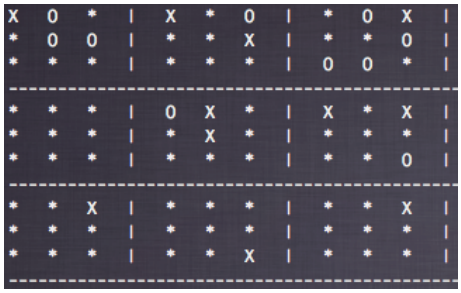

Figure 7: One example of 9Board

2.3 Structure of the program

The main structure of Advanced TTT is largely similar to part 1. I will mainly focus on the improvement and addtional part of Advanced TTT and will not repeat the overlap part with basic TTT. This game is a variant on basic TTT. Since every sub-grid is a basic TTT game. The win-loss is in determined by the same rules. However, 9 boards make the game more complicated to play, for AI, the depth of search much larger than previous one. The game also introduce the constraint : which grid to play on next?

2.3.1 Outline of the program

In this part, if a player has just played at some position on some board, then the next player must play on the board in the corresponding position in the grid. The example of several test case is shown below as Fig 8. The player who play the first step should choose which grid to play, then choose which step(S) to take within this grid. This is shown upper left of Fig 8. This step will force the next player to play in grid corresponding to the step S. This is shown upper right of Fig 8. Of course, the program will check the move it read are legal or not like part 1. This is shown lower left of Fig 8. In a situation that one grid is full, the program can inform the player that that grid is full, you could choose whatever grid you want to play. This is shown lower right of Fig 8.

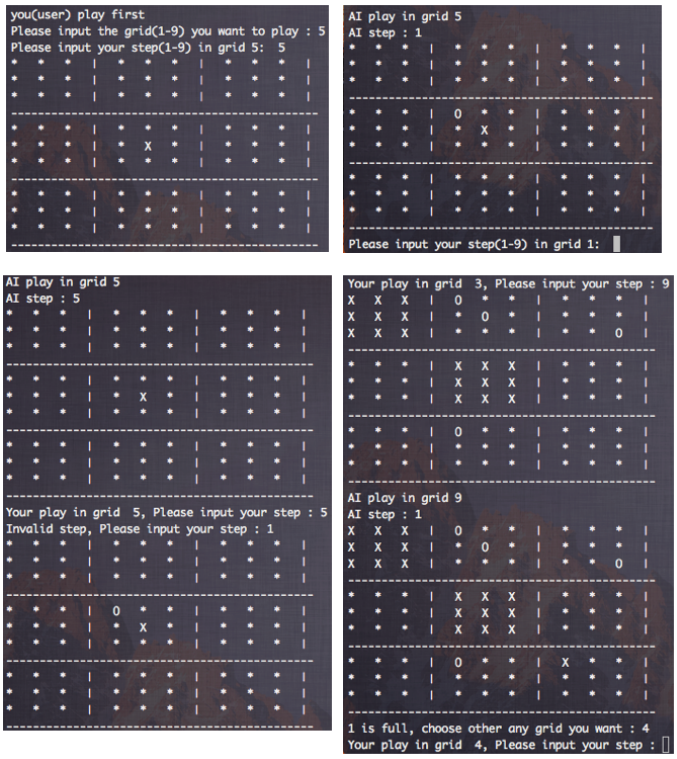

Figure 8: Structure test examples

2.3.2 Heuristic function implementation

The pure MINIMAX is infeasible for solving 9board program because depth of the program is mush higher compared to Basic TTT which is at most 9 steps. However, the step of 9board can be as much as 80. The factorial of 80 is a large number for personal computer. To overcome this problem, depth-limited-search(H-MINIMAX with a heuristic function) must be implemented.

My Heuristic function basically scan every line, row and diagonal three postion of the board, which means we scan 8 times each board.

During one scan, we count how many “X” and “O” in one line and return points according to this number of “X” and “O”. The points is given according to the following rules :

- 3 “X” and 0 “O”, return points 200*sign.
- 2 “X” and 0 “O”, return points 2*sign.
- 1 “X” and 0 “O”, return points 1*sign.
- 0 “X” and 3 “O”, return points -200*sign.
- 0 “X” and 2 “O”, return points -2*sign.
- 0 “X” and 1 “O”, return points -1*sign.

When AI play “X”, the sign = 1, when human plays “X”, the sign = -1. Sum the 8 times scan points gives the return of Heuristic function. In the MINIMAX, we also give 200 and -200 points if we the terminal test find either user has won the game. If we find 2 “X” and 0 “O”, I think there is possibility to win the game, so give the points 2. The examples of hueristic of several board is shown as Fig 9. This test can be done directly use method Heuristic() in Board class.

The heuristic function will be invoked if the depth is larger than the cutoff value. In the following example, the cutoff value we chooose is 8. The reason we chooose 8 is the tradeoff between the computation time and performance. Depth of 8 will take about 1 second for my PC get the results. And let AI “think” of depth 8 is already very strong.

One important point I need to make here is that we cannot use the same MINIMAX code we use in part1. We have to update the code to taking into account that the board to use give

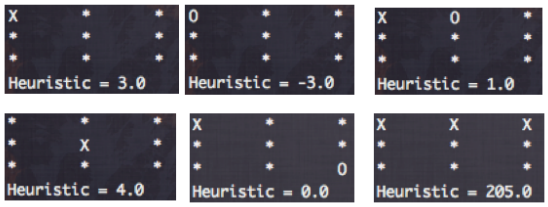

Figure 9: Heuristic performace test

hueristic value is changing all the time. We need to calculate the hueristic value every board along the path.

2.3.3 Heuristic function performance

To implement the heuristic function, we need to track the depth in the recursive MINIMAX algorithm. Once the depth larger than the cutoff, call heuristic function to get the estimated utility value. After all the description of heuristic function, let us test the performance of heuristic function through examples shown in Fig 10.

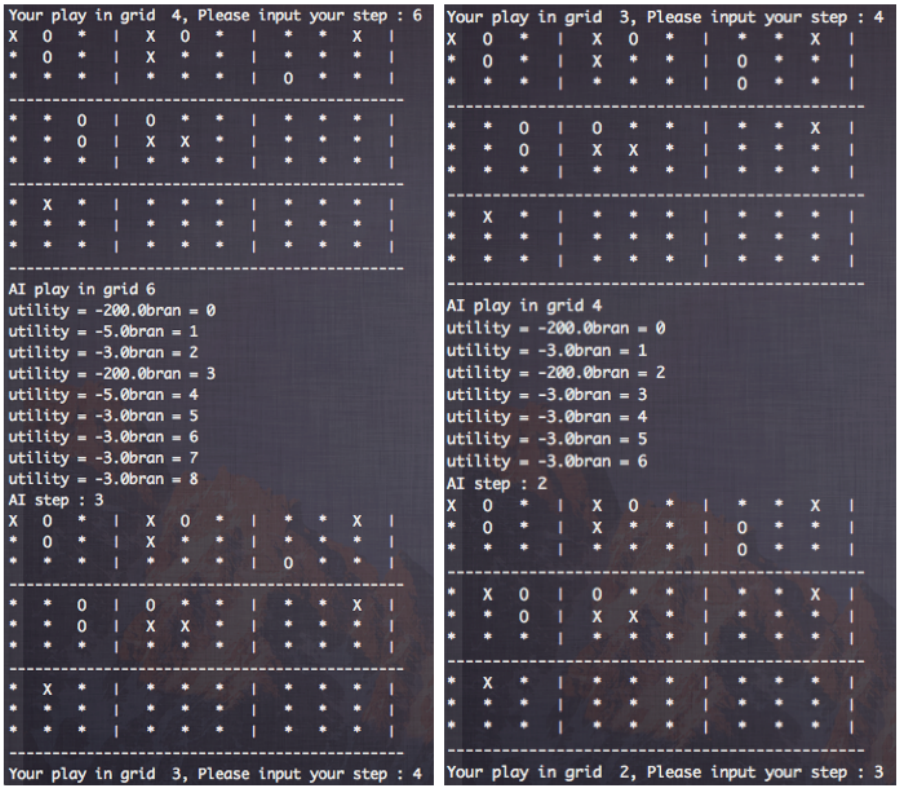

Figure 10: Heuristic performace test

The above figure shows the two steps during one game. AI play “X” and I play “O”. The right panel is right after the left panel. In the left panel I have to play in grid 4 because AI last step is 4. I choose to play at step 6. Then AI have to play at grid 6. The program print out the utilities to test whether the function is reasonable. Since grid 6 is empty as we can see, the AI have 9 options to choose, which are bran = 0 to bran = 8. The utility of taking step 1 (bran = 0) is -200. This is a correct judgment since there are two “O” and no “X” in column 2. If AI took step 1, this will let me play at grid 1 next step and I will win. The gives the utility value -200. Similar situation is at grid 4 in which I also have two “O” and no “X” in column 3. The AI corrected realize that cannot take step at 4 because utility will be -200. The AI choose step 3 because it is the maximum points(-3) it can get after 8 steps. Let us examime whether it is correct. The best step we could

do in grid 3 is taking step at 1, which will give the unility -3(-2+1-1-1). This is exactly what AI predicts. AI also predict the utility value at step 2 is -5. I did not test that manually since 8 depth is still too much for human like me. Based on the obvious results of -200, I am pretty confident that AI get correct utility values and make correct decision in this round of game.

After AI take step 3, leaving me to play at grid 3. I play at step 4(Not best move, just test AI). Then AI have to play at grid 4. AI have 7 choices at grid 4 since there are two grids have been occupied. AI corrected calculated that grid 1 and grid 4 is prohibitive for AI since It will let me win in next round since I have two two "O" and no "X" in these grids. The AI do not consider grid 3 because that point is occupied. AI finally take step 2 since the best I can do in grid 2 after 8 steps is -3. I cannot test that manualy since there are two many choices for me to count. However, the utility -200 is the one we can manually test. And it turns out the AI can correcly identify that.

2.3.4 Alpha-beta pruning

Alpha-beta pruning was used to eliminate large parts of the tree from consideration. The MINI-MAX algorithm is slightly updated to add the alpha-beta pruning. α and β is intiated as negative infinity and positive infinity. The running time of the "thinking" process of the program is monitored use stop watch from princeton algorithm developed by Robert Sedgewick and Kevin Wayne.

- effects on basic TTT : First let us compare how alpha-beta pruning affects the running time of basic TTT. The comparsion is shown as Fig 11. The left panel shows the running time of the case without alpha-beta pruning, the right panel include the alpha-beta pruning. We can clearly see that alpha-beta pruning significantly reduce the running time.

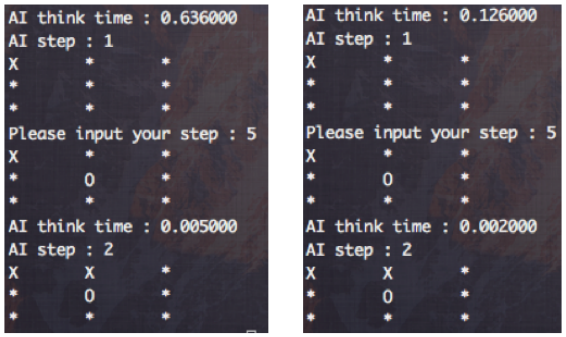

Figure 11: The effects of alpha-beta pruning on basic TTT

- effects on AdvancedTTT : To compare the difference, we set the cutoff of depth as 6. The comparison is plotted as Fig 12. The left panle is the running time without alpha-beta pruning and on right is with alpha-beta pruning. Both case is at the start 3 step of AdvancedTTT plat. We could also conclude that alpha-beta pruning greatly reduce the running time of the program.

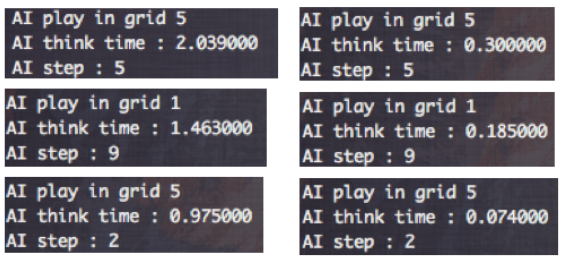

Figure 12: The effects of alpha-beta pruning on Advanced TTT

2.3.5 Program terminate test

To test whether the game is terminated, I count whether "X" or "O" appear in a line, just like what we do in part1 of the project. In part3 of the project. The terminal test must be changed to

let the game continue even if one grid is been win by “X” or “O”.

2.4 Program Performance Analysis : Will AI Lose?

After all the separate tests, let us play with our advancedTTT AI. The cutoff depth is set to be 8. I play with AI for several time but did not win a game. One example file(example.txt) that document all the step information is also attached. The file contain 39 steps. In the final step, I have five choice to play, but each of choice will lead the AI win in the round. The “think” time for AI at the beginning of the game is 2.301000s and reduce to 0.002000s at the end. If anyone interested in this example, you are very welcome to re-run the advancedTTT. Although I did not win the AI, it is possible to win AI in the case since we use hueristic at depth 8, which means AI did not take all the situations into account. But it is powerful enough to beat me with high possibility.

3 Super Tic-Tac-Toe

Comparing with Advanced Tic-Tac-Toe, Super Tic-Tac-Toe can use most of the Advanced Tic-Tac-Toe. The most important change we need to make is Updating terminate test.

3.1 Updating terminate test

To keep tracking the status of each sub-board of entire 9board, I add one array of length of 9 to represent the status of sub-board. For example, initial array elements to all zero. If board 1 is win by “X”, change the array[0] to 1, If board 1 is win by “X”, change the array[0] to -1. At the same time, mark all the element in the grid to “X” or “O”. The Fig 13 below shows the situation. The left panel shows grid 5 has been taken by “X”, the right panel shows grid 5 has been taken by “O”. The terminal test can be easily implemented by test the board status.

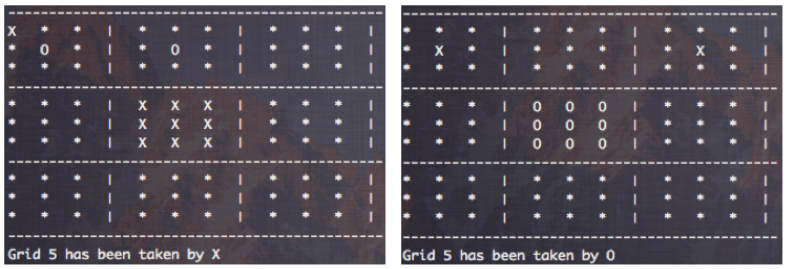

Figure 13: Grid has been win by X or O

Once we have the array to store the status of each sub-board. We can test whether 1 or -1 is in one line, column or diagonal of the board. Once the criterion is satisfied, the program will be terminated and inform use who is the winner. The winning status is shown as Fig 14.

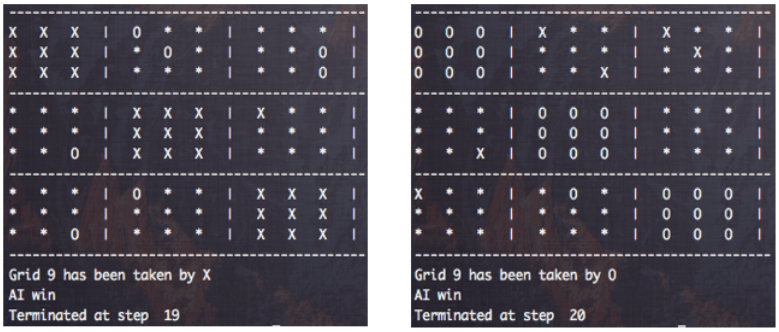

Figure 14: Final win status of X and O

3.2 Program Performance Analysis

I test the SuperTTT and write all the information to a file. I attached one example to the assignment. Following Fig 15 shows the final state, I won the game. AI is very close to win but behavir badly in the last few rounds. The whole process is documented. We can analyze the behavior of the AI from it. The overall step is 41.

Figure 15: Final win status of X and O