UNIVERSITY OF ROCHESTER

CSC 442: Project #1

Zhezheng Hong
Student ID: 31418089

September 30, 2018

Introduction

This document records all the work I have done to finish AI project one.

The solved problems include Basic Tic-tac-toe, Advanced Tic-tac-toe and Ultimate Tic-tac-toe

All these programs mentioned below are written in Java (version:10.0.1).

For more information, such as how to run and build these programs, please refer to the README file attached to each program separately.

This program is my own work based on my personal study and research. I have not worked with any other students and/or persons.

Part I: Basic Tic-tac-toe

Description of the problem

The rule of the basic tic-tac-toe is pretty simple. Player X and player O take turns to mark an open space in a 3x3 gird with their unique mark (X or O). The player who first succeeds in placing three of his/her mark in a horizontal, vertical or diagonal row wins the game.

Program design

For the basic tic-tac-toe, state-space search paradigm is applied.
I used the following variables to describe each component in the state-space search paradigm

States

- `game_state`: The state of this game. (playing/draw/X wins/Y wins)
- Board: A 3x3 char array used to represent the current state of the board.

Actions

- `TARGET_ROW`: `Row#` of this move;
- `TARGET_COL`: `Col#` of this move;
- `SEED`: Which seed (X or O) the player is holding

Transition Model

- Function UpdateState(action)

  Input:Action of a player

  Output: Update board according to this action

Initial State

- Function `Init_State()`

  Output: Empty board and reset game state to playing

Terminal State

- Function PlayerWon(): Check if someone wins
- Function IsDraw(): Check if it's a draw

  Input: board, move taken by the player

  Output: update the `game_state`: if someone wins or draw, terminate this game and start a new game

In this program, I created a class, namely PcPlayer, to represent the AI player. In this class I implemented two functions.

- PcPlayer(seed): Initialize the AI player object by assigning a seed
- MakeOptimalMove(state of the game): make an optimal move according to the current state of the game

The algorithm used to implement the function MakeOptimalMove() is **Minimax** algorithm. Every time this function is called, it will simulate the game by searching to the end of the game tree, assuming that the opponent will also play optimally. And it will assign a score to all the possible candidate moves. Finally, it will choose the move with the highest score as the optimal move and return it.

Analysis

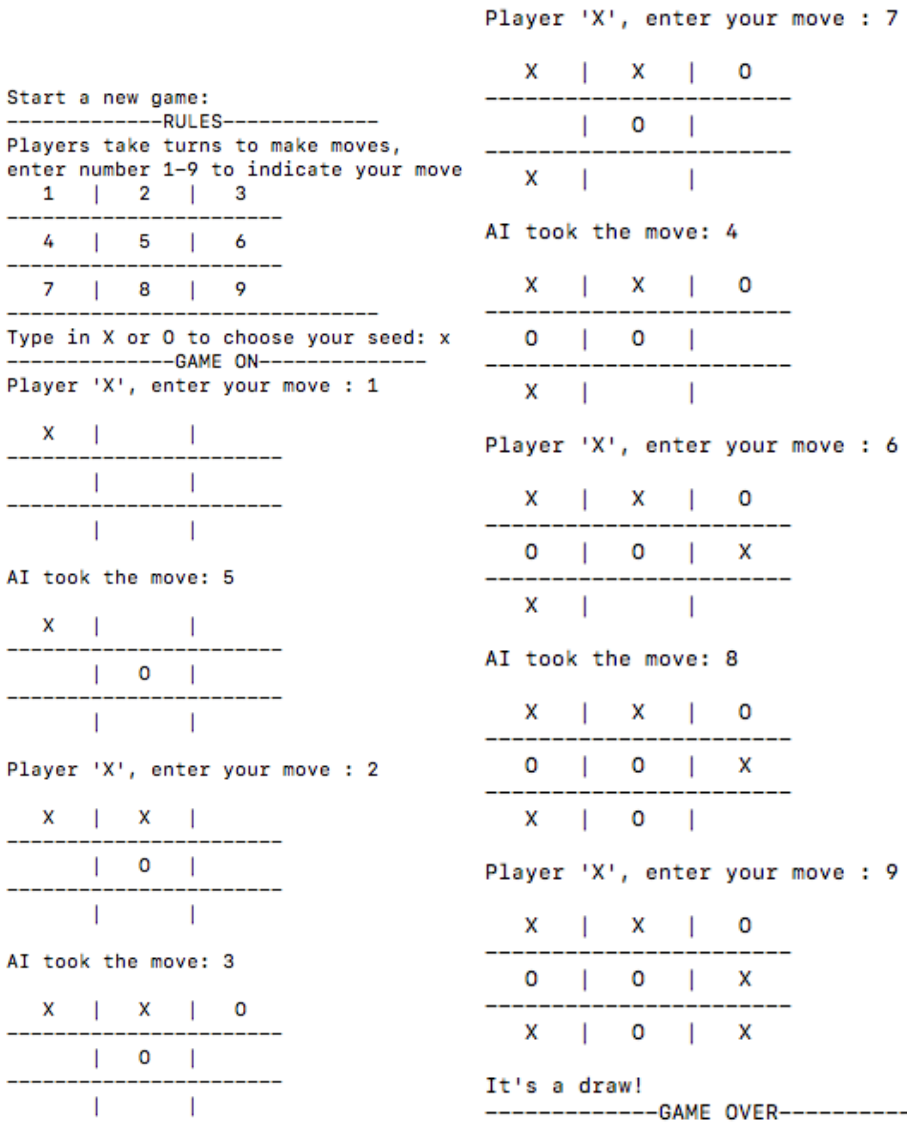
The Minimax algorithm is feasible for elementary problems like basic tic-tac-toe because the game tree is relatively shallow. In this case, the maximum depth of the game tree is 8 (start with depth 0).

This program should be able to either win the game or force a draw.

Performance evaluation

```
Start a new game:
-------------RULES-------------
Players take turns to make moves,
enter number 1-9 to indicate your move
   1   |   2   |   3
-----------------------
   4   |   5   |   6
-----------------------
   7   |   8   |   9
-------------------------------
Type in X or O to choose your seed: x
--------------GAME ON--------------
Player 'X', enter your move : 1

   X   |       |
-----------------------
       |       |
-----------------------
       |       |

AI took the move: 5

   X   |       |
-----------------------
       |   O   |
-----------------------
       |       |

Player 'X', enter your move : 2

   X   |   X   |
-----------------------
       |   O   |
-----------------------
       |       |

AI took the move: 3

   X   |   X   |   O
-----------------------
       |   O   |
-----------------------
       |       |

Player 'X', enter your move : 4

   X   |   X   |   O
-----------------------
   X   |   O   |
-----------------------
       |       |

AI took the move: 7

   X   |   X   |   O
-----------------------
   X   |   O   |
-----------------------
   O   |       |

Congratulation! Player O win!!
-------------GAME OVER-------------
```

This simulated game shows that the AI player could win the game if the human player doesn't play optimally.



This simulated game shows that the AI player plays the optimal move and can force the draw if the human player also play optimally.

Conclusion

This program works. The AI player always plays the optimal move. The time it takes for the AI player to make a choice of its move is trivial.

Part II: Advanced Tic-tac-toe

Description of the problem

Advanced tic-tac-toe is a variant on the basic tic-tac-toe. Instead of playing on a 3x3 grid, we have to play on nine 3x3 TTT boards which are arranged in 3x3 grid.

There is an important constraint for this game: 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.

No player should ever violates the above constraint. The player who first succeeds in placing three of his/her mark in a horizontal, vertical or diagonal row on a 3x3 local board wins the game.

Program design

The framework of this program is based on the program mentioned in part I. That means it also follows the state-space search paradigm.

However, the plain Minimax algorithm is infeasible to solve this problem because the depth of the game tree is so deep that it will take the Minimax algorithm incredibly long time to come up with an optimal move.

Therefore, to solve this program, I applied two techniques for adversarial search: depth-limited search and alpha-beta pruning.

- **Alpha-beta pruning**

  The idea of alpha-beta pruning is pretty simple. I added two input variables alpha and beta to the input of the Minimax algorithm. Alpha denotes the score of the best move for max (default value `INT_MIN`) and beta denotes the score of the best move for min(default value `INT_MAX`). As the searching process goes on, we use alpha and beta to evaluate the necessity of searching a new sub game tree. We either update the value of alpha or beta or abandon searching this sub game tree. In this way, we can significantly shorten the total amount of searching time.

- **Depth-limited search (Heuristic function)**

  The idea of Minimax algorithm is basically the same as depth-first search. In some complex searching spaces, such as the problem of advanced Tic-tac-toe, we cannot look all the way down to the bottom of the game tree. Therefore, in order to make the search possible, we must stop searching at some point of the game tree. In other words, we have to stop at a maximum depth. That means we cannot know the final result of taking this path so we have to come up with some evaluation or heuristic functions to evaluate the outcome of taking this path.

  For this program, this is the heuristic function that I came up with:

  We know that if both players play reasonably, they will always try to win the game or stop his/her opponent from winning the game. Therefore, the less open spaces on the board, the more likely that the game will end up with a draw.

According to the above analysis, I set a maximum searching depth for the Minimax algorithm. Every time a search path reaches the maximum depth, we stop searching and consider this move as a mediocre move that might lead to a draw. So we assign 0 to the score of this move. In other words, this move is not an optimal move for both players.

So basically the computer will start playing randomly at first. And as the game goes on, at some point when the limited depth allows the computer to search to the end of the game tree, it will start playing optimally.

Performance evaluation

```
Start a new game:
Type in X or O to choose your seed: x
-------------GAME ON-------------
Player 'X', enter your move : 5 5
Player may now place your mark on board: 5

       |      |             |      |             |      |
----------------------- ----------------------- -----------------------
       |      |             |      |             |      |
----------------------- ----------------------- -----------------------
       |      |             |      |             |      |

       |      |             |      |             |      |
----------------------- ----------------------- -----------------------
       |      |             |  X   |             |      |
----------------------- ----------------------- -----------------------
       |      |             |      |             |      |

       |      |             |      |             |      |
----------------------- ----------------------- -----------------------
       |      |             |      |             |      |
----------------------- ----------------------- -----------------------
       |      |             |      |             |      |


AI took the move: 5 1
Player may now place your mark on board: 1

       |      |             |      |             |      |
----------------------- ----------------------- -----------------------
       |      |             |      |             |      |
----------------------- ----------------------- -----------------------
       |      |             |      |             |      |

       |      |          O  |      |             |      |
----------------------- ----------------------- -----------------------
       |      |             |  X   |             |      |
----------------------- ----------------------- -----------------------
       |      |             |      |             |      |

       |      |             |      |             |      |
----------------------- ----------------------- -----------------------
       |      |             |      |             |      |
----------------------- ----------------------- -----------------------
       |      |             |      |             |      |
```

```
Player 'X', enter your move : 1 5
Player may now place your mark on board: 5
```


```
AI took the move: 5 2
Player may now place your mark on board: 2
```

```
Player 'X', enter your move : 2 5
Player may now place your mark on board: 5

       |       |               |       |               |       |
----------------------- ----------------------- -----------------------
       |   X   |               |   X   |               |       |
----------------------- ----------------------- -----------------------
       |       |               |       |               |       |

       |       |          O    |   O   |               |       |
----------------------- ----------------------- -----------------------
       |       |               |   X   |               |       |
----------------------- ----------------------- -----------------------
       |       |               |       |               |       |

       |       |               |       |               |       |
----------------------- ----------------------- -----------------------
       |       |               |       |               |       |
----------------------- ----------------------- -----------------------
       |       |               |       |               |       |


AI took the move: 5 3
Player may now place your mark on board: 3

       |       |               |       |               |       |
----------------------- ----------------------- -----------------------
       |   X   |               |   X   |               |       |
----------------------- ----------------------- -----------------------
       |       |               |       |               |       |

       |       |          O    |   O   |    O          |       |
----------------------- ----------------------- -----------------------
       |       |               |   X   |               |       |
----------------------- ----------------------- -----------------------
       |       |               |       |               |       |

       |       |               |       |               |       |
----------------------- ----------------------- -----------------------
       |       |               |       |               |       |
----------------------- ----------------------- -----------------------
       |       |               |       |               |       |


Congratulation! Player O win!!
-------------GAME OVER-------------
```

This simulated game shows that Computer will play under the rule. And it can win the game if human player doesn't play optimally. The runtime of this algorithm depends on the maximum depth limit we set.

We can make two computers play against each other and get more simulated game record. But for the neat of this document, I am not going to show that here.

Evaluation of the effect of alpha-beta pruning

To evaluate the effect of alpha-beta pruning, I simply compared the run time of two Minimax algorithms under the same circumstances(same maximum depth and same human player moves). One of them used the alpha-beta pruning while the other one did not. It turned out that with alpha-beta pruning, the time it took for the Minimax algorithm to come up with a solution is trivial (No more than 0.274 second). But without alpha-beta pruning, it took a significantly longer time for the algorithm to run. (I waited for 10mins before I gave up).

The result of this small experiment shows that alpha-beta pruning significantly improves the runtime efficiency of Minimax algorithm..

Part III: Ultimate Tic-tac-toe

Description of the problem

The game of ultimate tic-tac-toe basically has the same rule as the advanced tic-tac-toe mentioned in part II. The only difference is the winning condition. In order to win the ultimate tic-tac-toe, player has to win 3 horizontal, vertical or diagonal local boards.

Program design

This program is based on the implementation of part II. What I did was that I tweaked the final winning condition so that the player has to win three boards in a horizontal, vertical or diagonal row instead of winning any single board.

In the online version of ultimate tic-tac-toe, players can keep placing their marks on a board that is already won by one of the two players. In this way, a strategy for the player X could be applied to win the game by forcing player O to play on a certain board. But according to the requirement of this Project, if a board is already won by one player, then no more moves will be allowed on it. This rule makes the above strategy infeasible.

For the Minimax algorithm, some elementary heuristic functions, such as taking into account the number of local victories, might help improve the efficiency of the algorithm. But they are also not efficient enough.

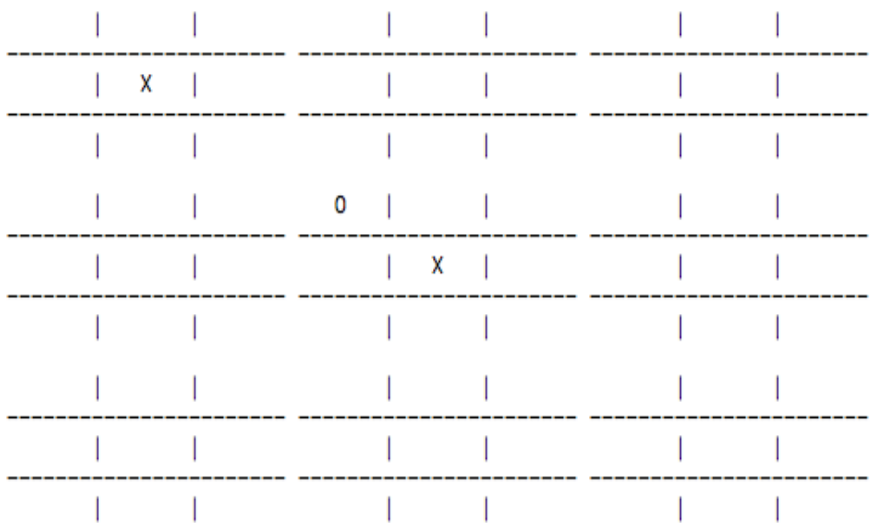
Monte-Carlo Tree Search method could be applied for this problem. But due to the limited amount of time for this project, I haven't implemented it.

Performance evaluation

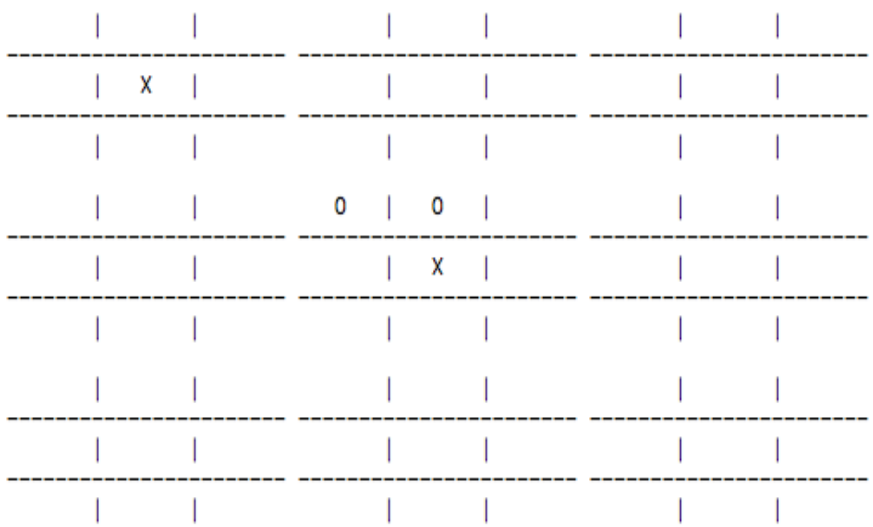
```
Start a new game:
Type in X or O to choose your seed: x
--------------GAME ON--------------
Player 'X', enter your move : 5 5
Player may now place your mark on board: 5
```


```
AI took the move: 5 1
Player may now place your mark on board: 1
```

```
Player 'X', enter your move : 1 5
Player may now place your mark on board: 5
```




```
AI took the move: 5 2
Player may now place your mark on board: 2
```



```
Player 'X', enter your move : 2 5
Player may now place your mark on board: 5

       |      |              |      |              |      |
----------------------- ----------------------- -----------------------
       |   X  |              |   X  |              |      |
----------------------- ----------------------- -----------------------
       |      |              |      |              |      |

       |      |          O   |   O  |              |      |
----------------------- ----------------------- -----------------------
       |      |              |   X  |              |      |
----------------------- ----------------------- -----------------------
       |      |              |      |              |      |

       |      |              |      |              |      |
----------------------- ----------------------- -----------------------
       |      |              |      |              |      |
----------------------- ----------------------- -----------------------
       |      |              |      |              |      |


AI took the move: 5 3
Player may now place your mark on board: 3

       |      |              |      |              |      |
----------------------- ----------------------- -----------------------
       |   X  |              |   X  |              |      |
----------------------- ----------------------- -----------------------
       |      |              |      |              |      |

       |      |          O   |   O  |   O          |      |
----------------------- ----------------------- -----------------------
       |      |              |   X  |              |      |
----------------------- ----------------------- -----------------------
       |      |              |      |              |      |

       |      |              |      |              |      |
----------------------- ----------------------- -----------------------
       |      |              |      |              |      |
----------------------- ----------------------- -----------------------
       |      |              |      |              |      |
```

```
Player 'X', enter your move : 3 5
Player may now place your mark on board: 5

       |       |               |       |               |       |
---------------------- ---------------------- ----------------------
       |   X   |               |   X   |               |   X   |
---------------------- ---------------------- ----------------------
       |       |               |       |               |       |

       |       |         0     |   0   |   0           |       |
---------------------- ---------------------- ----------------------
       |       |               |   X   |               |       |
---------------------- ---------------------- ----------------------
       |       |               |       |               |       |

       |       |               |       |               |       |
---------------------- ---------------------- ----------------------
       |       |               |       |               |       |
---------------------- ---------------------- ----------------------
       |       |               |       |               |       |


AI took the move: 1 1
Player may now place your mark on board: 1

   0   |       |               |       |               |       |
---------------------- ---------------------- ----------------------
       |   X   |               |   X   |               |   X   |
---------------------- ---------------------- ----------------------
       |       |               |       |               |       |

       |       |         0     |   0   |   0           |       |
---------------------- ---------------------- ----------------------
       |       |               |   X   |               |       |
---------------------- ---------------------- ----------------------
       |       |               |       |               |       |

       |       |               |       |               |       |
---------------------- ---------------------- ----------------------
       |       |               |       |               |       |
---------------------- ---------------------- ----------------------
       |       |               |       |               |       |
```

This simulated game shows that computer will play under the rule. But just like it's very difficult to find an effective evaluation function for the ultimate Tic-tac-toe, we cannot tell whether the computer's move is a good move or not. The runtime of this algorithm depends on the maximum depth limit we set for it to search.

— END —

Thanks for your time spent on grading this project. Have a nice day!