CS 5100 Spring Assignment 1
Tic-Tac-Toe Game Documentation
Rui Zhang NUID: 001989420

1. Game Design Decision

As we all know Tic-Tac-Toe is classic two-player turn-based game. So, we will have 2 players in the game and the game is played on a 3x3 board.
We can simply think of using object-oriented design to implement the game because the objects in this game are really clear to figure out. Therefore, in my implementation, I created two basic classes *Board* and *Player*. The *Board* defines the properties and behaviors of the board, and *Player* defines the players' properties and behaviors. However, we are building an AI agent who is supposed to paly against the human player. So, I also created a class called *AIPlayer*, which extends the basic *Player* class, and beside the basic actions a player can do, *AIPlayer* has some "thinking" behavior. The *AIPlayer* can estimate the current state of the board to figure out its best move, which will lead to a win or at least a draw.
Also, we need a controller to initialize the game and render information along the game. This controller is defined as a TicTacToe. It has run() and play() method, which are sufficient to conduct a game.

2. AI Agent Implementation Design Decision

The AI Agent is built to beat the human player or other agent from external environment, so the gold of the agent, to win, is conflicting with human player or other agent. In such a competitive environment, using adversarial algorithms to find a strategy to win the game seems the best fit for the Tic-Tac-Toe problem.

3. Adversarial Search Algorithm Choice

There are a lot of outstanding and smart adversarial search algorithms introduced in the book and online, however, I used MiniMax algorithm to implement the "thinking" behavior of the AI agent. Because the Tic-Tac-Toe unlike other combinatorial games, its possible games are small and even the game tree to search is relatively small. So, MiniMax algorithm can handle the "thinking" behavior easily and the running time is good enough to play the game.

4. Minimax Algorithm Implementation

In the code, the "think then act" behavior is defined by AIPlayer. evaluateAndMarkCell().

evaluateAndMarkCell() method consumes a board object and read its current state, evaluate current state and find out the best move, then execute the best move. The best move is decided by the method minimaxDecision(). Every time the board is given to the AI after a human player's move, the AI use this method to evaluate the game and make its decision.

minimaxDecision() computes the minimax decision from the current state. It consumes a board object, and get the current state of the board from the board, found out all possible moves in current state of the board. Then compute the minimax value of each move by recursively compute the minimax value all the way down to the leaves of the game tree. After all minimax values of the successor state of current state are computed, get the maximized minimax value and use the move corresponding to the best minimax value as the best move.

minValue() and maxValue() are two mutual-recursive methods, they two together finish the recursion process mentioned above. minValue() can be explained as the AI player, it evaluate all possible minimax values result from all its possible moves, tries to minimize the possible loss. And the maxValue() can be explained as a opponent that the AI is trying to beat, it tries to maximize the possible gain. The AI recursively evaluate the whole game tree to find out the best next move by using these two methods. When it reaches to the bottom of the game tree, the algorithm returns the utility value of the final state. If the final state is a win, utility is 1, lose is -1 and draw is 0. The utility will be backed up to upper recursion call to help decide the minimax value of upper tree node.

5. Analysis of Minimax Algorithm

Because the game here is a two-player turn-based game, and two-player play against each other, the adversarial search algorithms are best solutions. Minimax is one of those algorithms and it's especially suitable for the environment that is fully observable and deterministic. The environment where Minimax suits must be competitive as well. In general, for any game that the game is a two-player turn-based and the game is a combinatorial game with finite possible games and finite game tree, we can use Minimax to implement the AI agent for it.