TicTacToe (3×3) AI

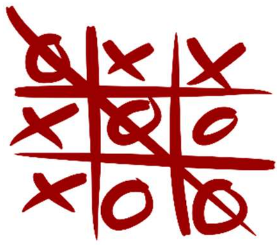
Preamble

This assignment introduces you to a core Artificial Intelligence (AI) concept that utilizes trees. In this assignment, you will implement a tree that can approximate the best possible move given a game board. Please read the whole handout before starting as there are many new concepts you have not seen before.

Learning Objectives

- Achieve proficiency with using trees and recursion
- Learn about the minimax algorithm
- Develop an awareness for how computationally infeasible it may be to enumerate all possible paths of a game

TicTacToe: The Game



The purpose of this project is to create an AI program that can skillfully play TicTacToe. Tic-tac-toe is a board game for two players, X and O, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a diagonal, horizontal, or vertical row is the winner.
(Source: https://en.wikipedia.org/wiki/Tic-tac-toe)

Your job

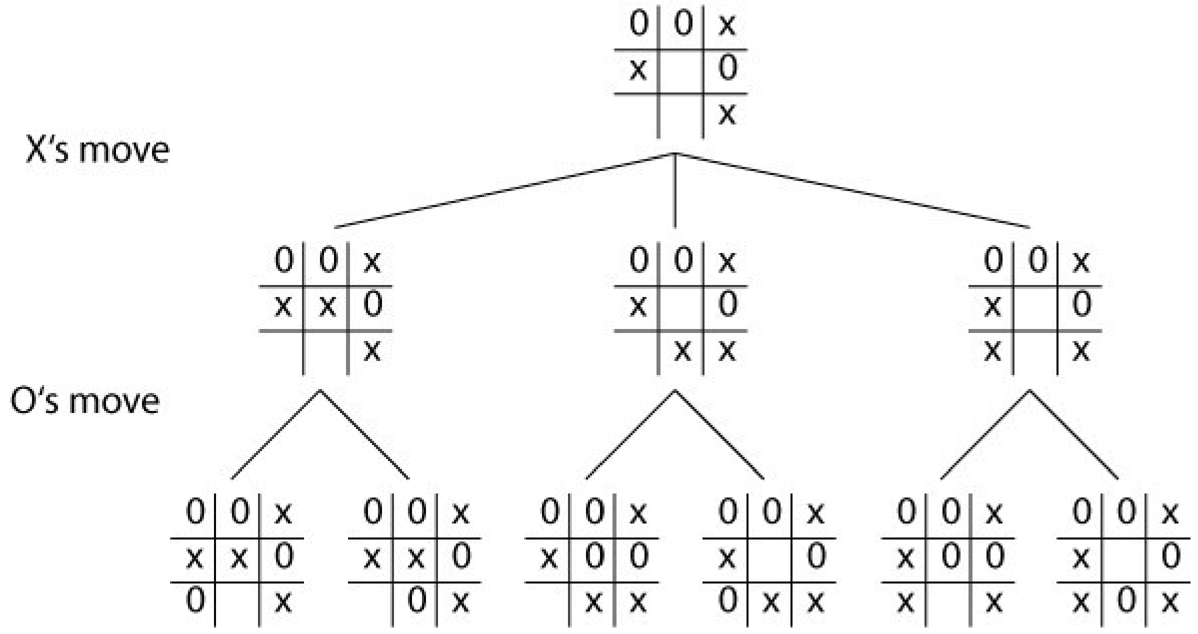
There are two parts to this assignment.

1. You have to develop a TicTacToe game that a human player can play against a simple AI player (random). The game states can be represented in command line or graphic user interface.
2. You have to write an AI program (minimax) to play against human players.

Minimax Algorithm

Suppose your program wants to make a move in TicTacToe. Before it makes the move, it will look ahead at the possible sequences of moves it and the opponent could make and determine which first move is best, based on all those sequences of moves. The key data structure behind this is a tree in which the nodes are possible game states. Each state consists of the current board and which player's turn it is. From this state, the AI can generate all possible future states that can result after that player takes his turn.

Below is a partial game tree for TicTacToe. In the root of the tree, Player X has three possible places to play, so there are three states children of the root state. From each of those states, the states are then generated for Player O's potential moves. This can continue until a state is reached that ends the game or results in a tie game.

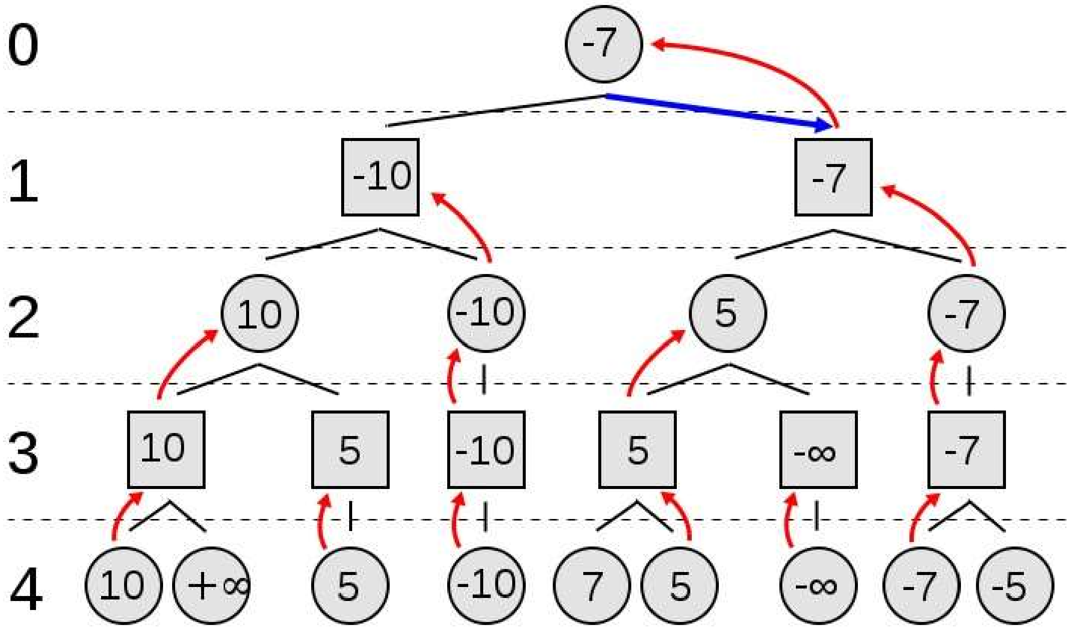


The last thing to do is evaluate the parent nodes in the tree. The game state tree creator (you) is trying to win the game. You want to maximize your score, while your enemy is trying to minimize your score.

This knowledge will help determine the value for the parent states. If the current state is your move (i.e. it is your turn in that state), use the *maximum* value from the list of children as the current state's value. If the current state is not your move, use the *minimum* value from the list of children.

Now you see why it is called the Minimax algorithm.

For example in the illustration below, the circles represent your color and squares the opponent. The tree has already been constructed and the evaluation function was run at level 4. It is the opponent's turn at level 3, so the minimum value is always selected from the children. While at level 2, it is your turn, so you always select the maximum value from level 2's children. Ultimately, the best possible known move bubbles up to the top where at level 0, the maximum value state is selected as the next move.


0
1
2
3
4
-7
-10
-7
10
-10
5
-7
10
5
-10
5
-∞
-7
10
+∞
5
-10
7
5
-∞
-7
-5