Minimax Algorithm

Bruno Porto Masquio

10/11/2015

What it is

Description

The minimax is an algorithm in IA used to play specific games. It uses the fact that each player are against each other and wants to do the best moves to win in the end. With this information, it is possible to make predictions about which future states can be reached from a specific point of the game.

Having this resource, the computer will do the best moves in order to maximize its chances of winning and minimizes the chances of an opponent win.

Requirements

Turn-based: The game must be based in turns which each player can only make a move in his own turn.

Full information available: This means that each player knows the whole scenario of the game and there is with no hidden information. A player is able to know every possible move that his opponent can do. For instance, scrabble does not apply to this requirement because the players doesn't have information about the other's hands, which makes impossible to predict the opponent's moves.

Logic-based: Game must be based in rules and premises that are able to describe the current state of the game and the next possible moves.

Applications

The minimax algorithm can be used in various games, such as tic-tac-toe, checkers, othello, and chess.

How it works

This algorithm creates a tree structure with the following configuration.

Nodes: Represents a state of the game.

Edge: The sons of node A are the ones that represent possible states after one move.

Then, given a specific configuration of the problem, it will expands the tree, evaluating every possibility. Considering that every player will take the best moves, it goes through the path that will end in the best result for the player.

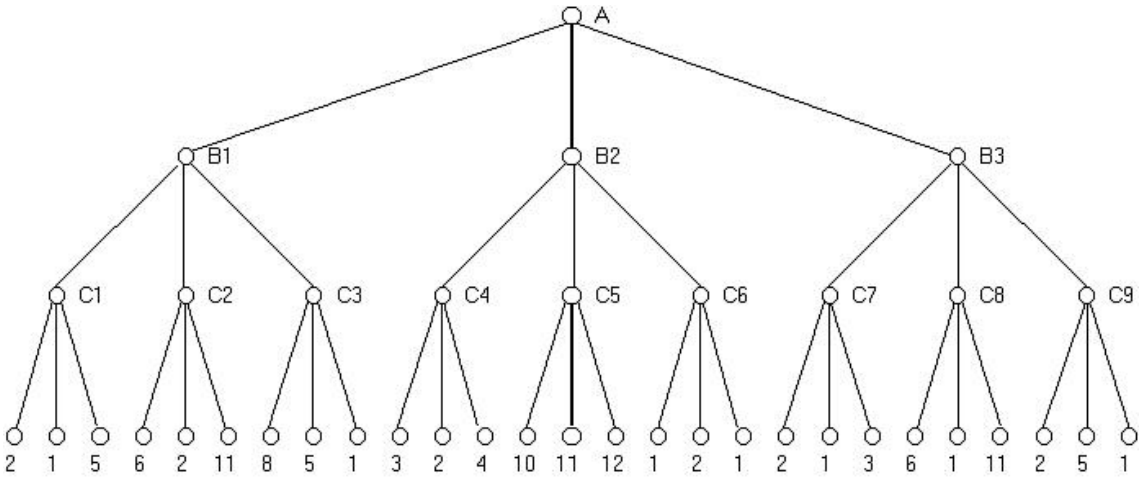

Figure 1. An example of a game tree with the playing possibilities.

We can imagine this tree above as a game with the Minimax requirements. Let's consider that it is the computers turn at level A, level B will be the humans, level C the computers turn and so on. The leaf nodes are the end of the game and the score related to it.

If the computer plays in a level, he will go for the greatest values possible of its node sons. In the algorithm, this would be treated as the maximum levels. In the example, they are A and C levels. So, the C1 value will be 5, C2, 11, C3, 8, and so on.

If it is the humans turn, the algorithm consider that it will make the best move, which will chose the lowest value of its sons. In this example, it is the B level. So, the B1 value will be the lowest of C1, C2 and C3, that is 5. B2 will be 2 and B3 will be 3.

Now, knowing the best move that the human can do in his turn(B), the best decision of the A turn is to find again the greatest value of B1, B2 and B3. In this case will be 5 by B1.

So, this algorithm is based in searching the maximum or the minimum values in specific levels. It is observed that we will choose it considering who is playing in the turn. This makes the algorithm always go for the best state.

Example in a real game

We have an example below of the tic-tac-toe and how does the Minimax algorithm would see this current state of the game.

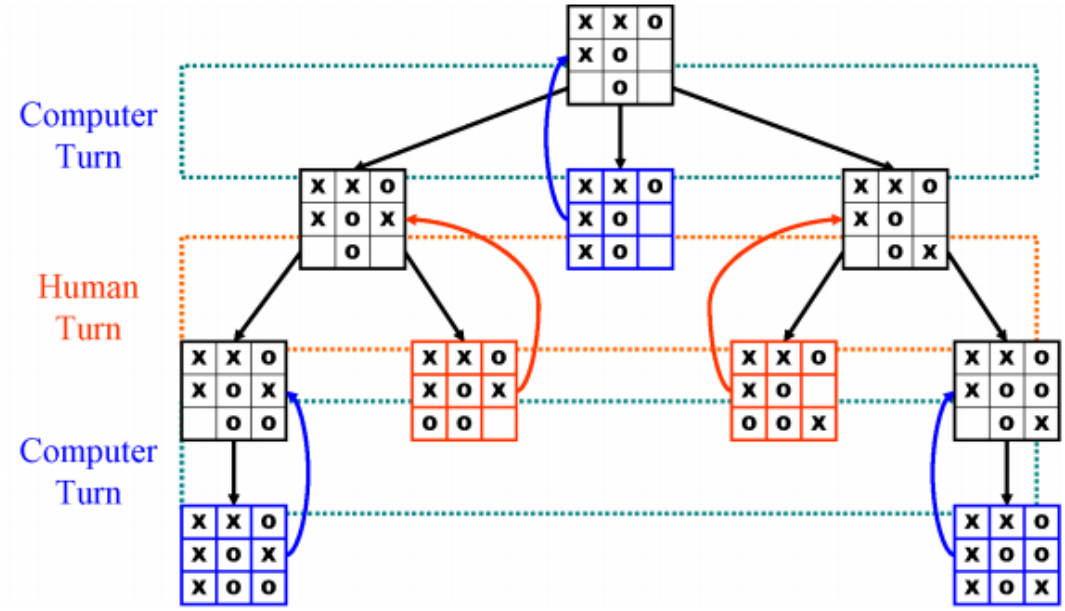

Figure 2. A state of the game tic-tac-toe and its tree representing the possible moves that can be taken. The computer is playing as 'X' and the human is playing as 'O'. The blue boards represents that the computer wins while the orange represents humans victory.

In this particular point of the game(root node), the computer is able to do 3 moves. The first and the third, even if there is a possibility of wining, this depends on a bad move of the human. To be safe, we assume the opponent is brilliant and will pick the best possible move for himself, which will make these two moves not ideal. So, the best move is the second one, which always makes the computer wins.