Minimax in Arbitrarily board-size of Tic-tac-toe

By: Daniel Guzman

CSU Chico State

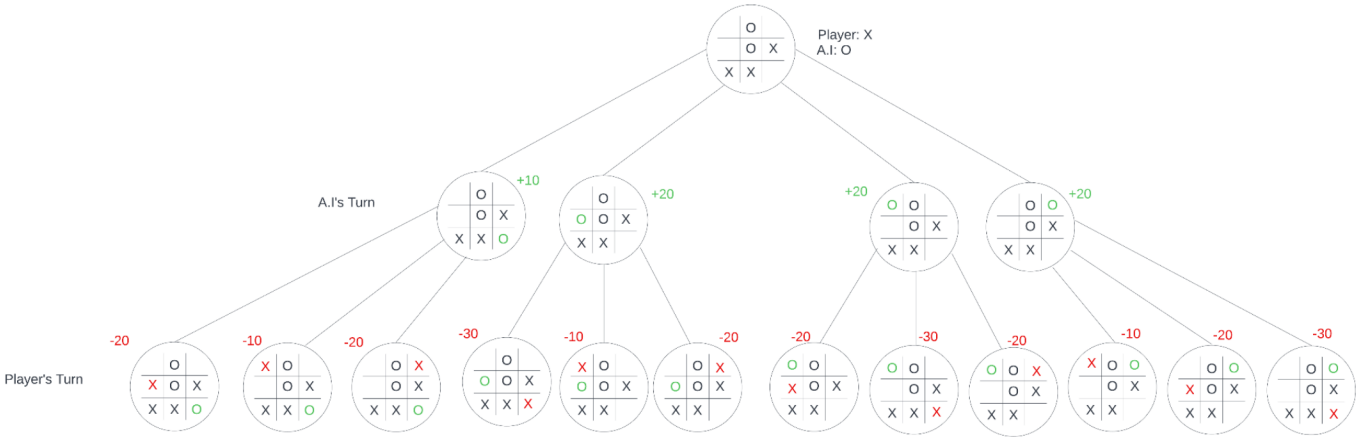
3/25/2022

The Minimax algorithm developed and proven by John Von Neumann is considered the basis on the subject of Game Theory and for developing an A.I model / CPU for any turn based game, where the objective of the A.I opponent is to Minimize or Maximize the gains of its decisions without explicitly knowing the next decision of other players playing the game. The general idea of The Minimax algorithm is that it is designed in a worst-case approach, dare I say the brute force approach, such that in order to determine the best action to take against all other players in the game, the algorithm will check / calculate the gains of all possible actions of the other players to determine which move to take.

An example would be, say that you are playing a game of chess with an A.I that is modeled by the Minimax algorithm, in this scenario your objectives would be to play the game normally and determine which moves would be best to beat the opponent, on the other hand, the A.I's objectives is to calculate the state of the gameboard given all possible moves that the A.i can make, and evaluate which gameboard state results in the best outcome that will minimize the gains of all other players in the game.

The way we should think about this algorithm conceptually is as a Decision tree where the root is the state of the gameboard after the A.i has placed their piece in a given position, given how many more positions are available from that state is how many states will branch off of that root state. From there, the A.i will then continue to branch off of game board states until it has reached a terminating state in all branches such that either the player or A.i has won or the game board is in a Draw state. The returning value after reaching the terminated state should be the best score along the branches from each state.

The follow image represents a decision tree created by the Minimax algorithm when used on a Tic Tac Toe Game board state:



Following the image we can see that the first initial state inserted in the Minimax algorithm is the root node, since there are 4 available positions the Minimax algorithm can recursively branch off of the root node and to the next game board states, where then each child node now has the next player's inserted pseudo piece. From there the game board state is evaluated and given a score based on the game's heuristic, if the game board evaluates to a victory or a draw then that node doesn't branch and the score is returned back to the parent node. The final result should be the best move that the A.i can make to win the game.

For this project, to showcase my implementation of the Minimax algorithm, I implemented the algorithm in the board game of Tic Tac Toe written in the language C++. Additionally, I designed the Minimax algorithm to work with an arbitrary-sized Tic Tac Toe board such that a player can play against an A.i at a game of Tic Tac Toe of board-size greater than 3x3.

The following is the pseudo code of the Minimax algorithm that was implemented as a helper function in the process of choosing the optimal move for the A.i

```
function minimax(ismin, Board, symbol)
    score = evaluate(board, symbol)

    if (terminated_state(board))
        return score

    if (board.filled())
        return 0

    if (ismin)

        bestScore = INT_MAX

        for i in 0 to board.dimension
            for j in 0 to board.dimension
                if (board[i][j] == EMPTY)

                    board[i][j] = symbol

                    score = minimax(false, board, 'X')
                    bestScore = min(bestScore, score)

                    board[i][j] = EMPTY

        return bestScore

    else if (!ismin)

        bestScore = INT_MIN

        for i in 0 to board.dimension
            for j in 0 to board.dimension
                if (board[i][j] == EMPTY)

                    board[i][j] = symbol

                    score = minimax(true, board, 'O')
                    bestScore = max(bestScore, score)

                    board[i][j] = EMPTY

        return bestScore
```

We first evaluate the current board and set a variable called score which will be used as a temp variable to keep track of evaluation score of available position on the board of that current state. We then check to see if the board is filled or at a terminated state such that either the player or A.i got a "3 in the row". From here, we check to see who's turn it is within the decision tree by checking if *ismin* is true or false. If true then we check which empty position current left on the board yields the best score for that player. The same can be said if *ismin* is false. This form of the Minimax algorithm was used in a helper function called optimalMove() which was used as sort of a layer to find the optimal move given the at most current state of the board after the player has gone.

Now let's talk about the Run Time Analysis. For this implementation of the Minimax algorithm the Run Time will solely be based off of the amount of nodes/board-states that are made within the decision tree as that is the main component of the algorithm. Say that we begin with a state where there are 4 possible ways to make a move, similar to the example shown earlier, given that state the Minimax algorithm will then need to recursively call itself 4 times to "find out" which move yields the better score. Upon calling itself, it will then need to recursively call itself 3 more times from the 4 child states that it has just created in order to figure out which move is optimal from *that* state. It will continue doing so until it has reached a terminated state or the board is filled and no possible moves are left. Under those circumstances, unfortunately the Asymptotic Runtime will be $O(b^m)$ where b is the number of moves from the initial state and m is the absolute possible depth of the decision tree.

Reference

Introduction to Game Playing, CIS Temple. 2022, https://cis.temple.edu/~vasilis/Courses/CIS603/Lectures/l7.html. Accessed on 3/26/2022

Minimax Algorithm in Game Theory, GeeksforGeeks. 2022, https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/?ref=lbp. Accessed on 3/25/2022

Minimax, Wikipedia, 2022, https://en.wikipedia.org/wiki/Minimax. Accessed on 3/25/2022

Strategies of Play. CS Stanford, 2022, https://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/game-theory/Minimax.html. Accessed on 3/25/2022