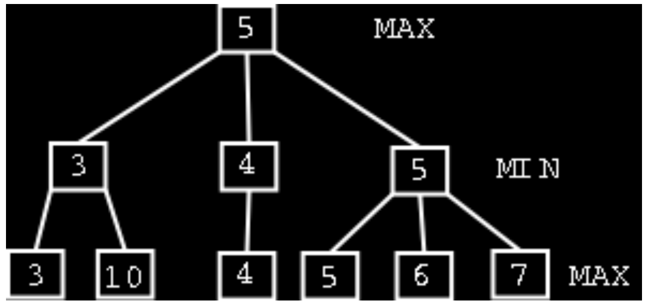
Search in Games

- Search is particular effective in *zero-sum* games - games where
  - *one player's loss is the other player's gain.*
- Most board games fall into this category:
  - Chess, checkers, tic-tac-toe, etc.
- An effective search strategy exists for these kind of games:
  - The *minimax* algorithm.

MiniMax

- The minimax algorithm attempts to compute the best next move from the perspective of one of the players
- It builds a search tree of the game state space
- And then traverses the tree *maximizing* the players chances to win
- However, the algorithm also takes the moves of the other player into account who tries to *minimize* the chances that the first player wins (hence the name)
- Key to this algorithm is an *evaluation function* that can score board configurations.

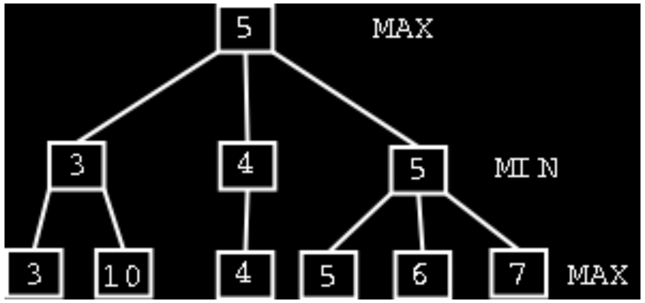


Best possible outcome for player 1, but *unreachable* because of the actions of player 2, would put player 2 into a favorable position.

The board position that is reachable with minimum damage

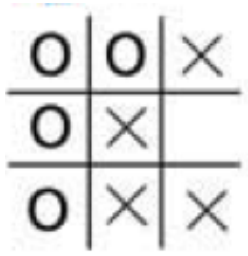
Minimax

```
minimax(player,board)
    if(game over in current board position)
        return score
    children = all legal moves for player from this board
    if(max's turn)
        return maximal score of calling minimax on all the children
    else (min's turn)
        return minimal score of calling minimax on all the children
```



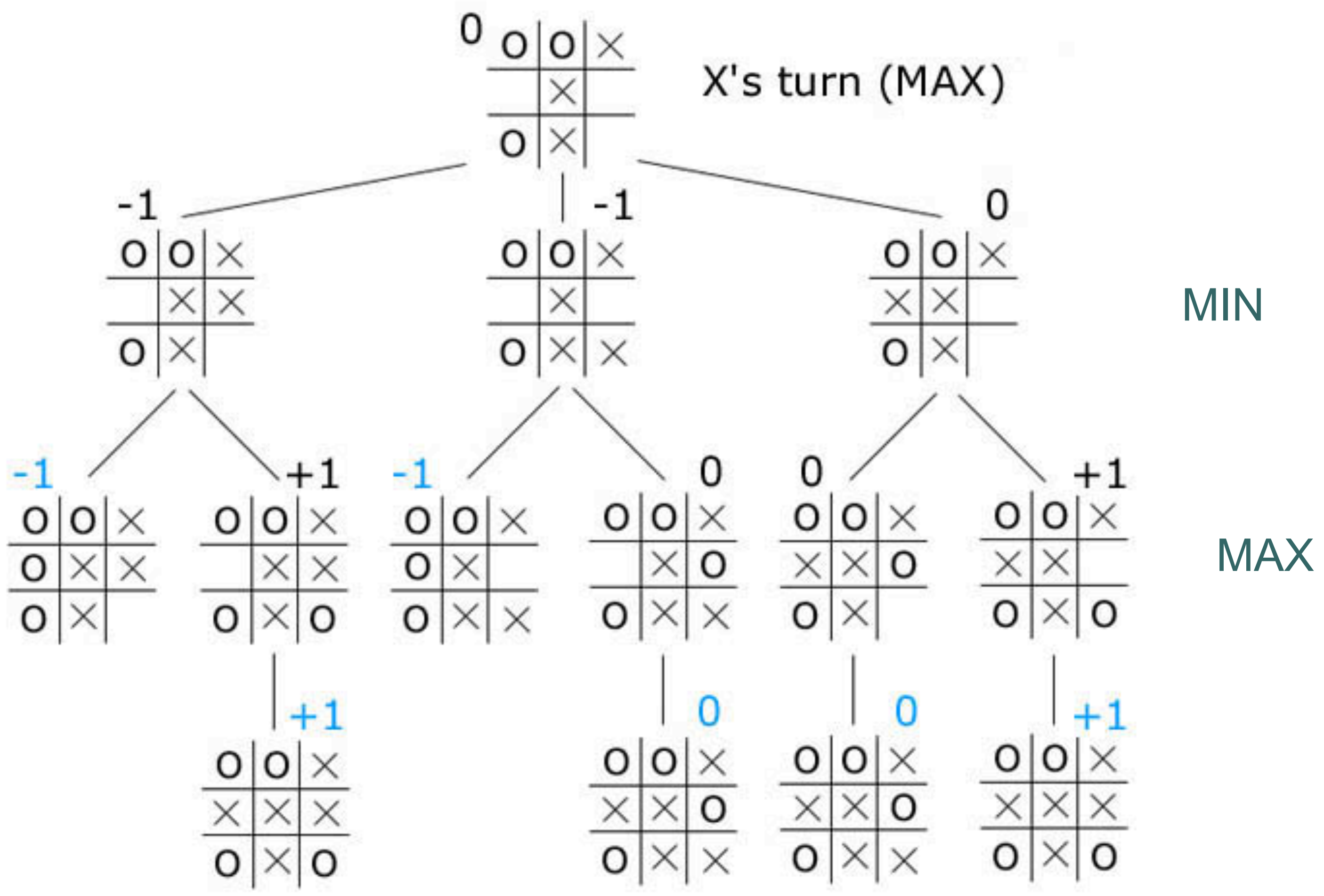
Tic-Tac-Toe

- Zero-sum, two player game
- Aim: capture either a row, column, or diagonal.
- Evaluation function:
  - Player1: X
  - Player2: O
  - Win for X = +1
  - Win for O = -1
  - Draw: 0



Score: -1

Tic-Tac-Toe



More Complex Games

- In more complex games like chess it is impossible to search the tree all the way to “terminal game states”
- Set a lookahead limit and search the tree to this limit using the evaluation function to tell us what the leaves of this abbreviated search look like.