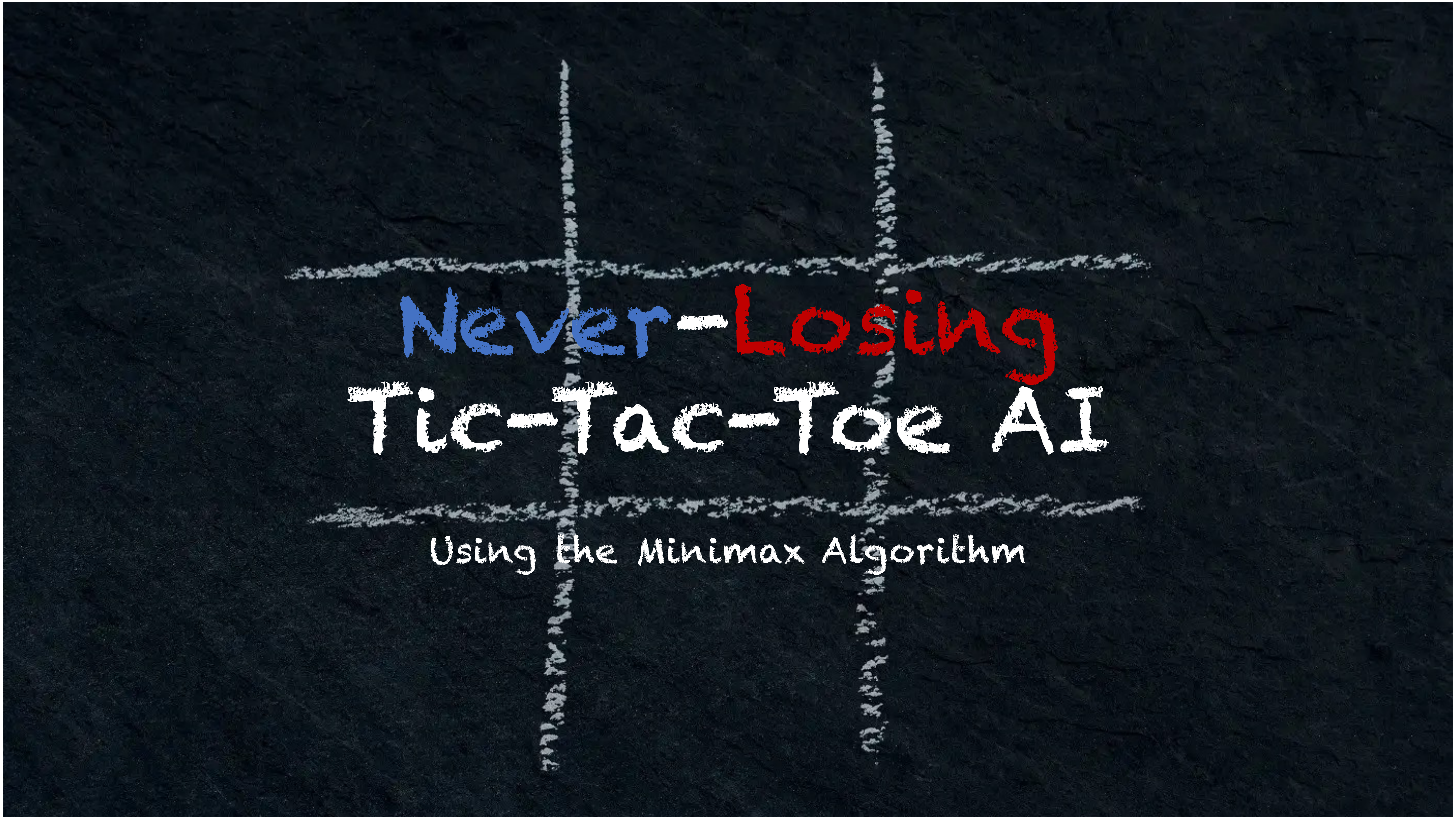
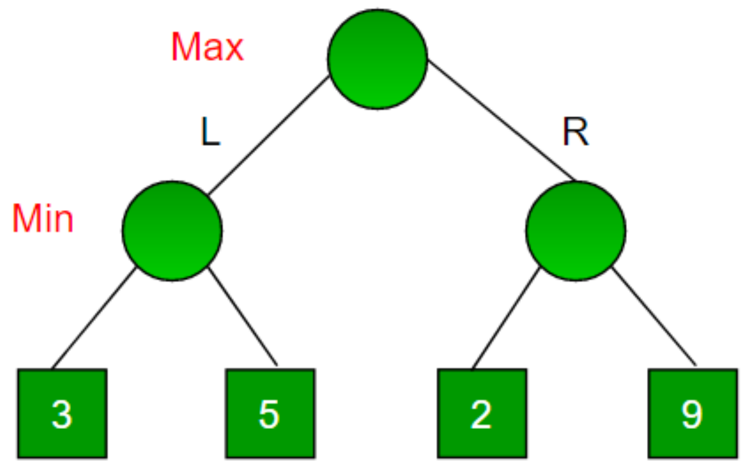


Never-Losing Tic-Tac-Toe AI

Using the Minimax Algorithm

MiniMax

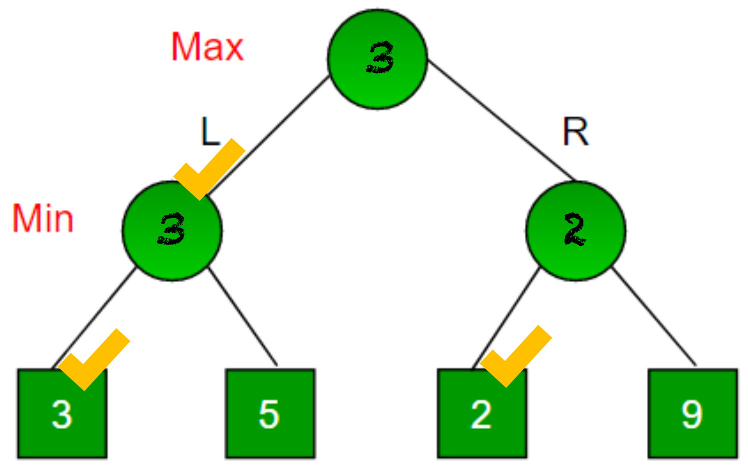


Suppose... On a turn-based game,
I want to get the highest possible score

1. I first choose the optimal choice for myself
2. My opponent then chooses their optimal choice

What choice should I make
to get the best possible outcome?

MiniMax



1. The best result for me is 9, which is in R
2. However, if I choose R, my opponent will surely choose 2
3. On the other hand,

if I choose L, my opponent will choose 3

So, the choice that will lead me to the best case is L, even though the best possible case is in R

We take turns choosing the best case for each other.
In my POV, my opponent's best choice is MIN, my best choice is MAX
Hence the name : MINIMAX Algorithm

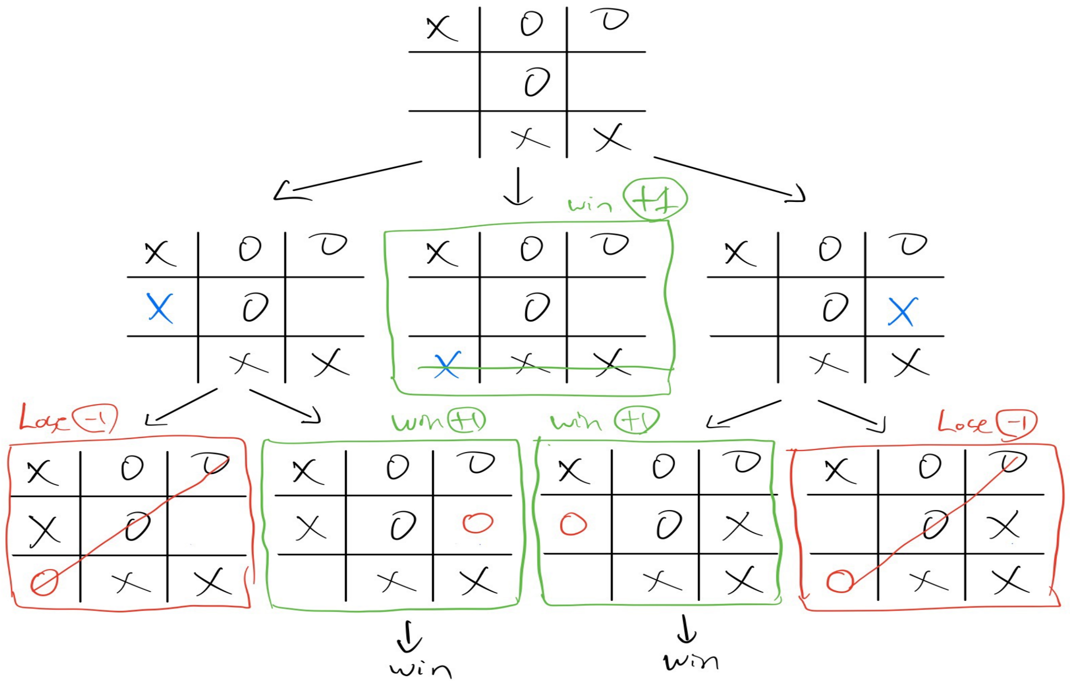
Get all possible results -> Backtrack best case -> Determine next step

MiniMax in Tic-Tac-Toe

Three possible results: Win:1, Lose:-1, Draw:0

Example:

(Mid-game)



*If the algorithm is called on an empty board, it will play out all possible outcomes of the whole game before choosing the next step

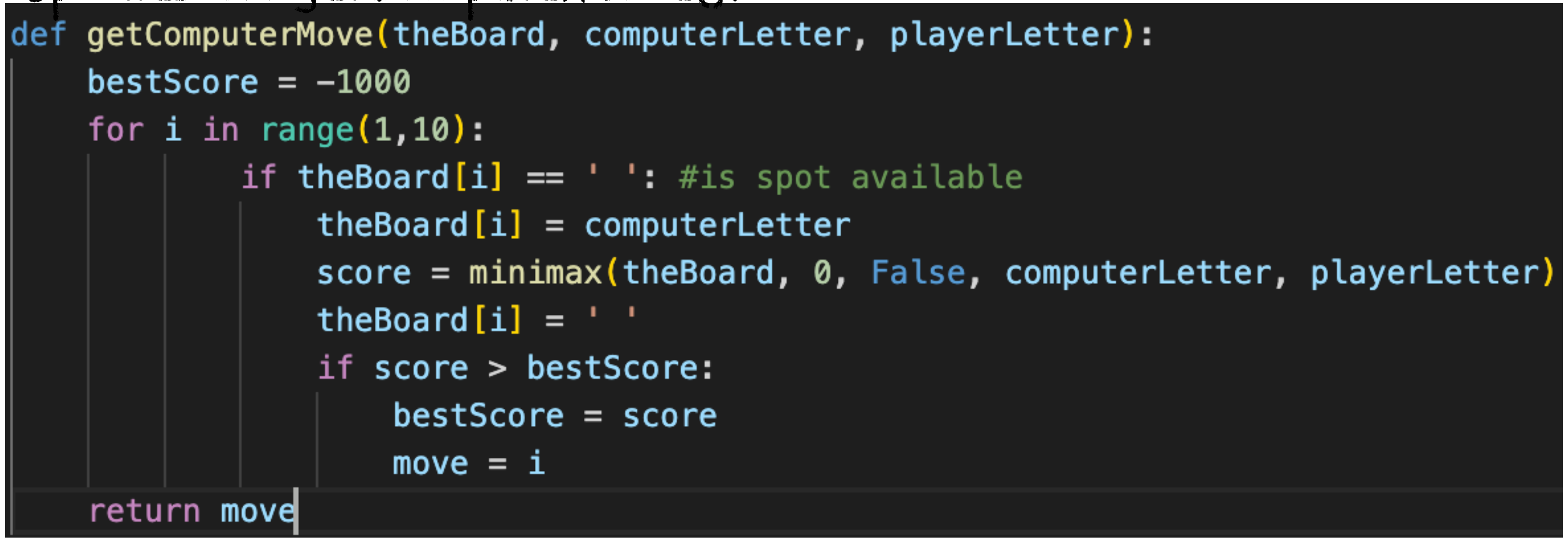
9! Leaves, Each -1|0|1

Get all possible results starting from current state -> Find the best cases -> Find the best path by backtracking -> Determine next step

Code Implementation

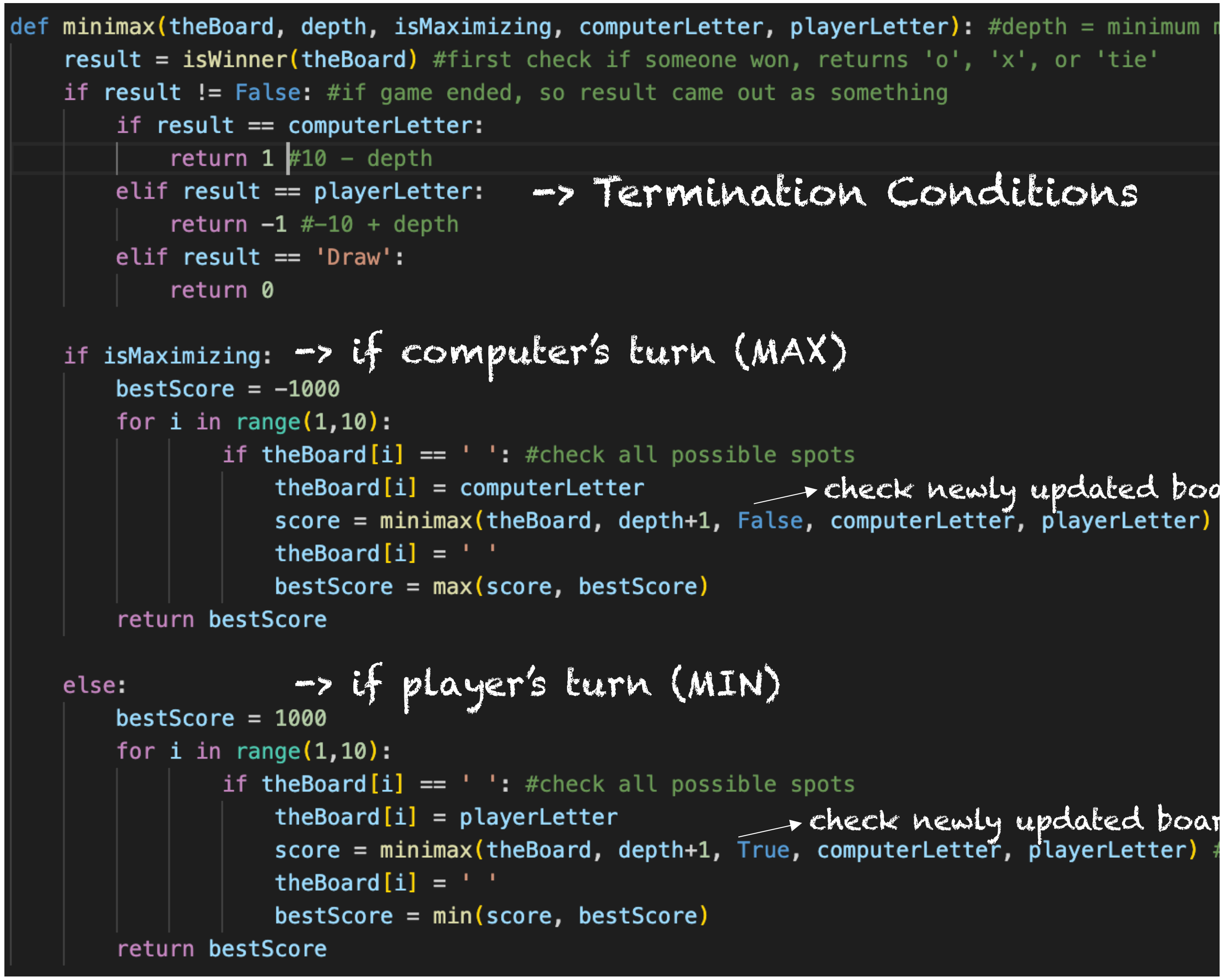
Since we have to get all end results first, then backtrack: Use <u>RECURSION</u>

Updates on getComputerMove():



```python
def getComputerMove(theBoard, computerLetter, playerLetter):
    bestScore = -1000
    for i in range(1,10):
            if theBoard[i] == ' ': #is spot available
                theBoard[i] = computerLetter
                score = minimax(theBoard, 0, False, computerLetter, playerLetter)
                theBoard[i] = ' '
                if score > bestScore:
                    bestScore = score
                    move = i
    return move
```

For every empty space on the board, put in the computer's letter and determine all results leading from that choice by calling minmax() on the updated board



```python
def minimax(theBoard, depth, isMaximizing, computerLetter, playerLetter): #depth = minimum
    result = isWinner(theBoard) #first check if someone won, returns 'o', 'x', or 'tie'
    if result != False: #if game ended, so result came out as something
        if result == computerLetter:
            return 1 #10 - depth
        elif result == playerLetter:
            return -1 #-10 + depth
        elif result == 'Draw':
            return 0

    if isMaximizing:
        bestScore = -1000
        for i in range(1,10):
                if theBoard[i] == ' ': #check all possible spots
                    theBoard[i] = computerLetter
                    score = minimax(theBoard, depth+1, False, computerLetter, playerLetter)
                    theBoard[i] = ' '
                    bestScore = max(score, bestScore)
        return bestScore

    else:
        bestScore = 1000
        for i in range(1,10):
                if theBoard[i] == ' ': #check all possible spots
                    theBoard[i] = playerLetter
                    score = minimax(theBoard, depth+1, True, computerLetter, playerLetter) #
                    theBoard[i] = ' '
                    bestScore = min(score, bestScore)
        return bestScore
```

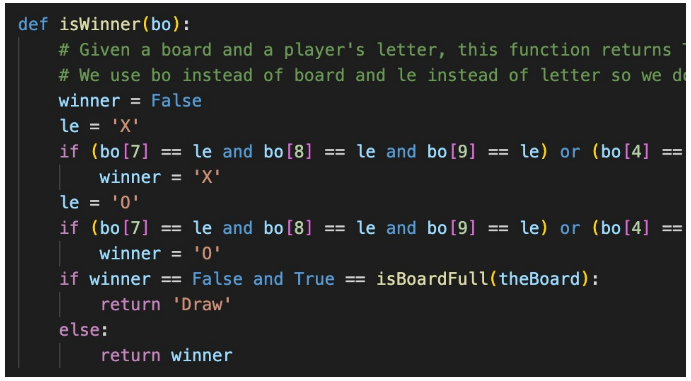
-> Termination Conditions

-> if computer's turn (MAX)

→ check newly updated board again, but now in MIN (player's turn)

-> if player's turn (MIN)

→ check newly updated board again, but now in MAX



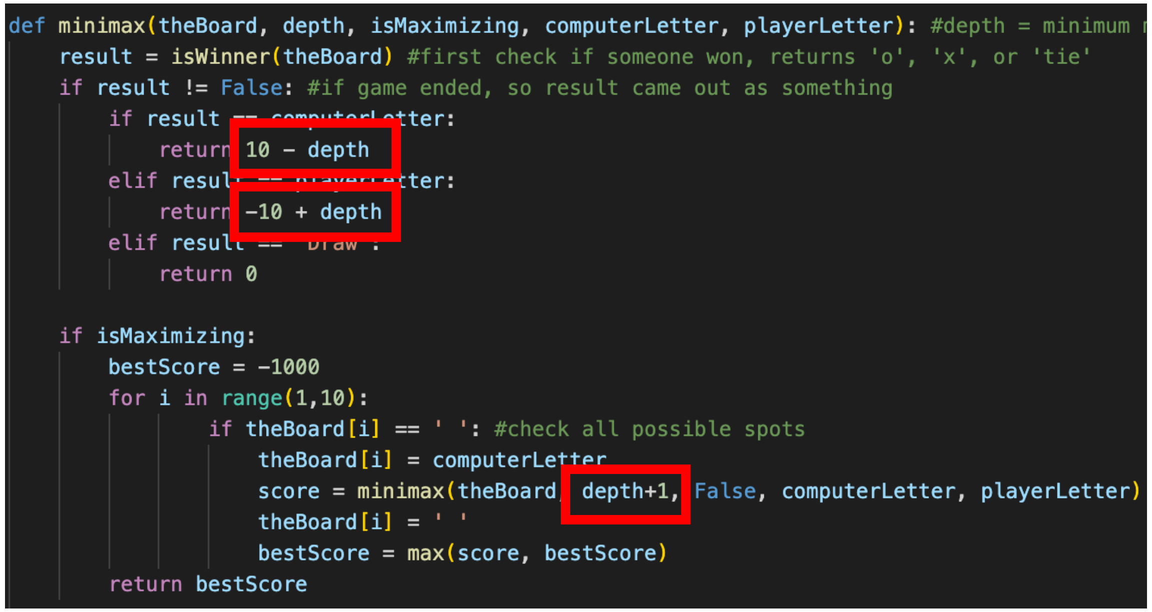
```python
def isWinner(bo):
    # Given a board and a player's letter, this function returns
    # We use bo instead of board and le instead of letter so we d
    winner = False
    le = 'X'
    if (bo[7] == le and bo[8] == le and bo[9] == le) or (bo[4] ==
        winner = 'X'
    le = 'O'
    if (bo[7] == le and bo[8] == le and bo[9] == le) or (bo[4] ==
        winner = 'O'
    if winner == False and True == isBoardFull(theBoard):
        return 'Draw'
    else:
        return winner
```

Improvements

If there are multiple choices that can lead to a win but each have different numbers of steps, it would be inefficient to prolong the game by choosing the one with more steps.

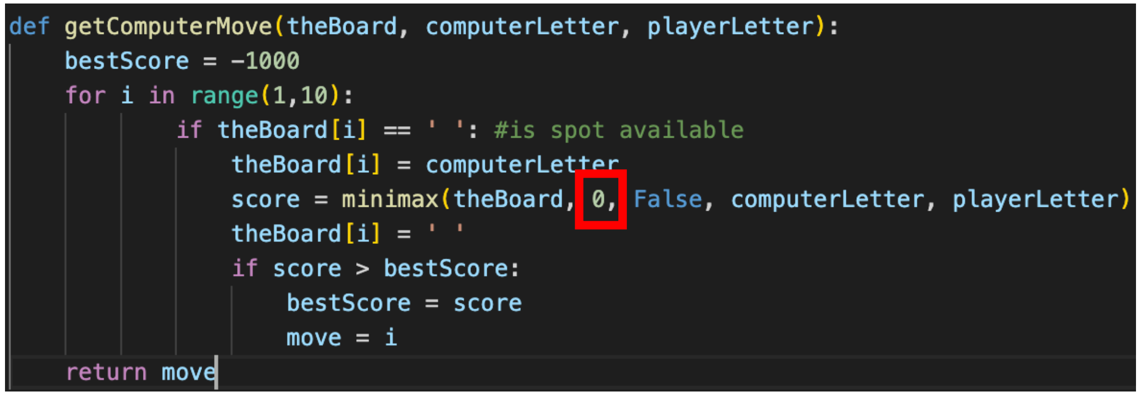
Use 'depth' :

- Increment in each minimax() call
- Depth represents number of plays
- lesser plays = better win



```python
def minimax(theBoard, depth, isMaximizing, computerLetter, playerLetter): #depth = minimum
    result = isWinner(theBoard) #first check if someone won, returns 'o', 'x', or 'tie'
    if result != False: #if game ended, so result came out as something
        if result == computerLetter:
            return 10 - depth
        elif result == playerLetter:
            return -10 + depth
        elif result == 'Draw':
            return 0

    if isMaximizing:
        bestScore = -1000
        for i in range(1,10):
            if theBoard[i] == ' ': #check all possible spots
                theBoard[i] = computerLetter
                score = minimax(theBoard, depth+1, False, computerLetter, playerLetter)
                theBoard[i] = ' '
                bestScore = max(score, bestScore)
        return bestScore
```



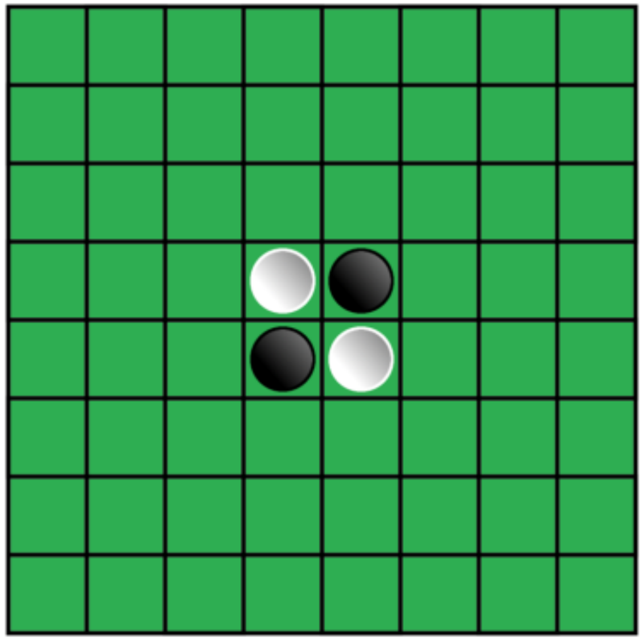
```python
def getComputerMove(theBoard, computerLetter, playerLetter):
    bestScore = -1000
    for i in range(1,10):
        if theBoard[i] == ' ': #is spot available
            theBoard[i] = computerLetter
            score = minimax(theBoard, 0, False, computerLetter, playerLetter)
            theBoard[i] = ' '
            if score > bestScore:
                bestScore = score
                move = i
    return move
```

The smaller the depth,
The bigger the return value

Since getComputerMove chooses max(),
We will automatically choose the result
with the smallest depth -> least plays

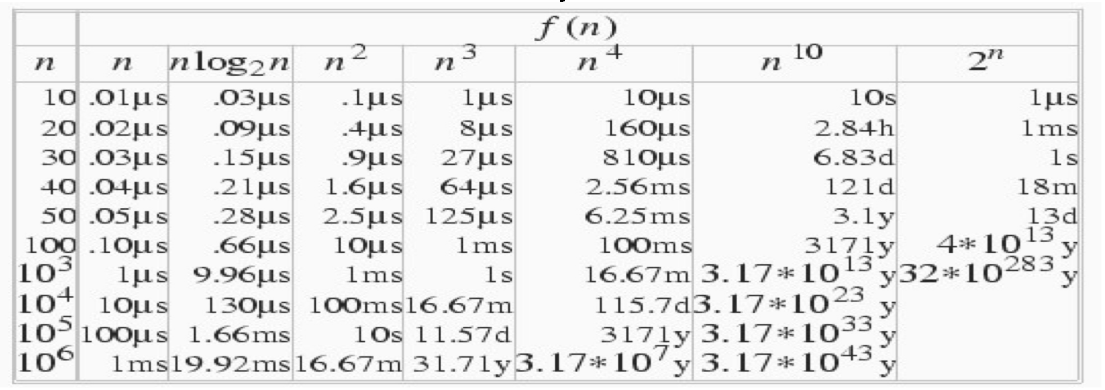
Return value is set to '10' - depth,
Because max depth is 9 (3x3 grids)
Value can be any number bigger than 10,
But it must be smaller than boundary num
(1000)

Reversi



Tic Tac Toe had (3x3)! Steps

Reversi has (8x8)! Steps

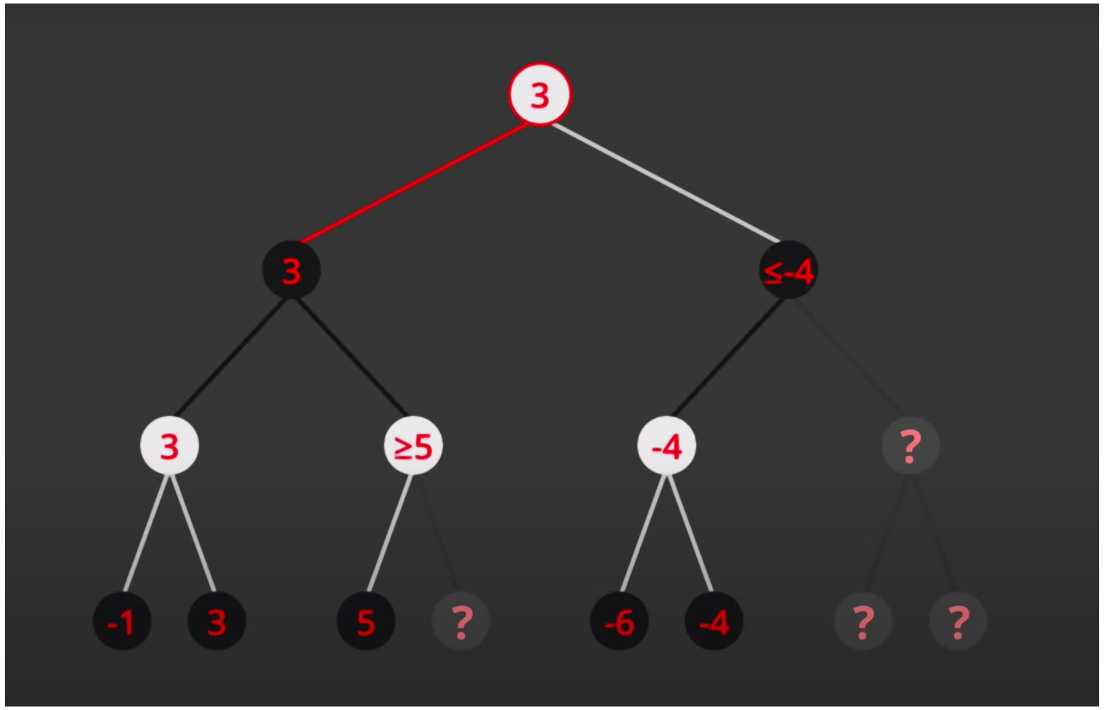


	$f(n)$						
n	n	$n\log_2 n$	n^2	n^3	n^4	n^{10}	2^n
10	.01μs	.03μs	.1μs	1μs	10μs	10s	1μs
20	.02μs	.09μs	.4μs	8μs	160μs	2.84h	1ms
30	.03μs	.15μs	.9μs	27μs	810μs	6.83d	1s
40	.04μs	.21μs	1.6μs	64μs	2.56ms	121d	18m
50	.05μs	.28μs	2.5μs	125μs	6.25ms	3.1y	13d
100	.10μs	.66μs	10μs	1ms	100ms	3171y	$4*10^{13}$ y
10^3	1μs	9.96μs	1ms	1s	16.67m	$3.17*10^{13}$ y	$32*10^{283}$ y
10^4	10μs	130μs	100ms	16.67m	115.7d	$3.17*10^{23}$ y	
10^5	100μs	1.66ms	10s	11.57d	3171y	$3.17*10^{33}$ y	
10^6	1ms	19.92ms	16.67m	31.71y	$3.17*10^{7}$ y	$3.17*10^{43}$ y	

Exec time on a 1 billion instructions per sec computer.

N! is even worse than 2^n, so 64! Will take a long time

Reversi - pruning



We know that the score we will get from LLL is 3. If we've found out that 5 is in LRL, Since white is going to choose the best outcome, whatever it chooses LR will be bigger than 5. However, Black is always going to choose whichever is smaller than 3, and LR. Therefore, we dont need to calculate LRR to know that the best case is 3.

Same goes for the left tree. Since black will choose the worst possible score, whatever it chooses will be smaller than -4. Since White is going to choose the best score, it the result will always be 3. There fore we dont need to compute the whole of LL

References

GeeksforGeeks. 2022. *Minimax Algorithm in Game Theory | Set 1 (Introduction) - GeeksforGeeks.* [online] Available at: <https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/> [Accessed 31 December 2022].

GeeksforGeeks. 2022. Introduction to Evaluation Function of Minimax Algorithm in Game Theory - *GeeksforGeeks.* [online] Available at: < https://www.geeksforgeeks.org/introduction-to-evaluation-function-of-minimax-algorithm-in-game-theory/> [Accessed 31 December 2022].

GeeksforGeeks. 2022. *Minimax Algorithm in Game Theory | Set 3 (Tic-Tac-Toe AI- Finding optimal move) - GeeksforGeeks.* [online] Available at: < https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-3-tic-tac-toe-ai-finding-optimal-move/> [Accessed 31 December 2022].

Shiffman, D. [The Coding Train]. (2019 Dec. 11). *Coding Challenge 154: Tic Tac Toe AI with Minimax Algorithm* [Video]. Youtube. https://www.youtube.com/watch?v=trKjYdBASyQ

Sahni, S. *Data Structures, Algorithms, And Applications in C++,* Silicon Pr, 2004 Jan. 1