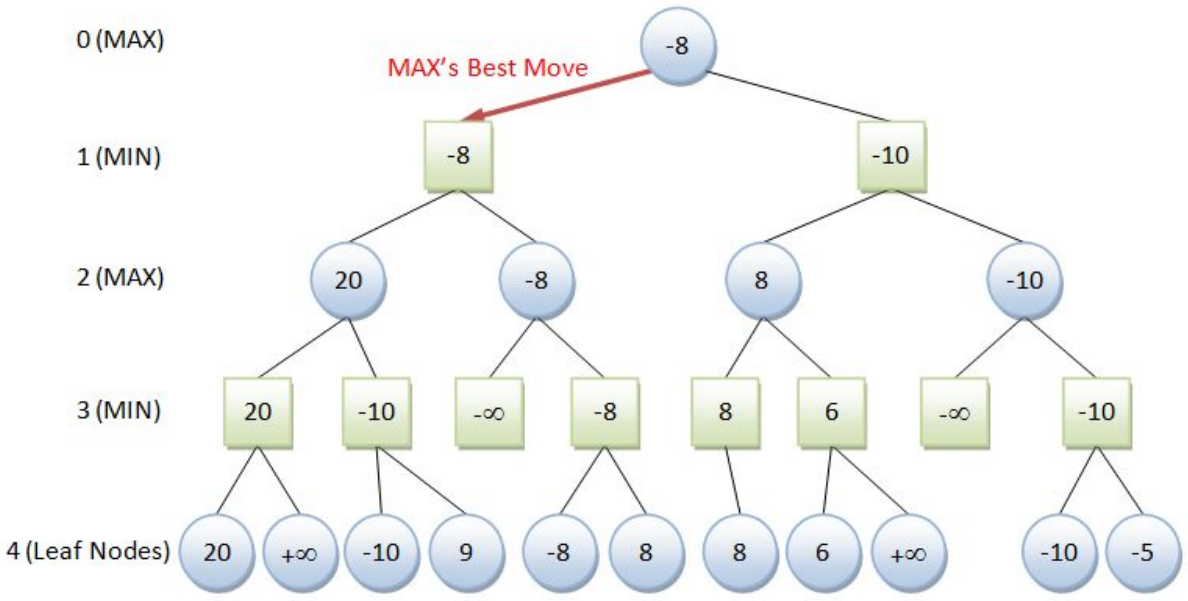
MINIMAX ALGORITHM

Definition –

Given that two players are playing a game optimally (playing to win), MiniMax algorithm tells you what is the best move that a player should pick at any state of the game. So, the input to MiniMax algorithm would be –

1. State of the game.
2. Whose turn it is.

And the output would be the best move that can be played by the player given in the input.



Idea of MiniMax Algorithm –

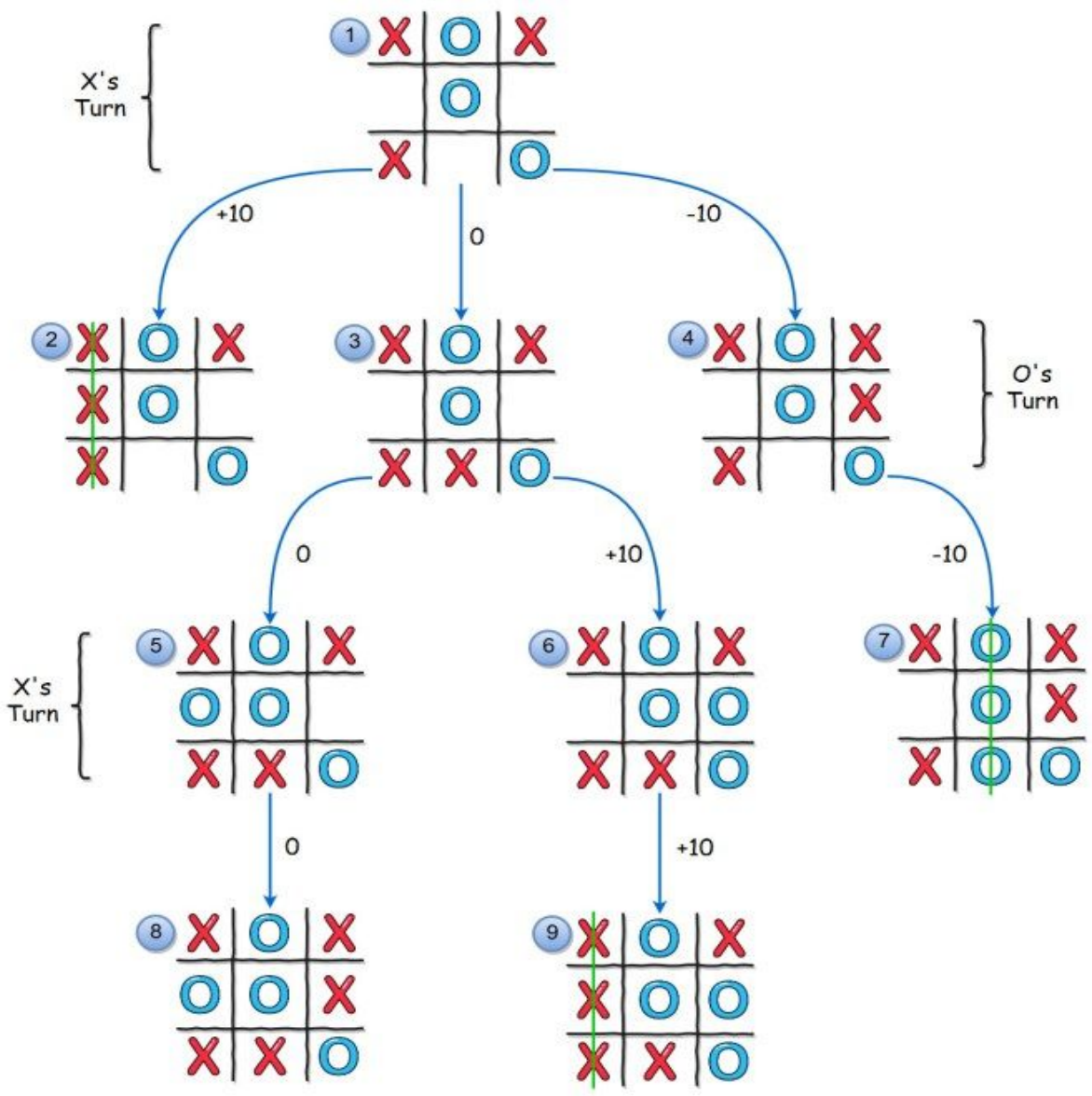
In the game of Tic-Tac-Toe, there are two players, player X and player O. Now imagine there's a scoreboard which displays a huge number called "score", and –

1. If X wins, the score increases by 10.
2. If O wins, the score is decreased by 10.

3. If it is a draw, then the score remains unchanged.

So now, the bigger the number score has, the better it is for X, which means X will try to maximize score as much as possible. Similarly, the lesser the score, the better it is for O, so O will try to minimize the score as much as possible.

To understand this better, imagine you are player X and this is the current state of your game –



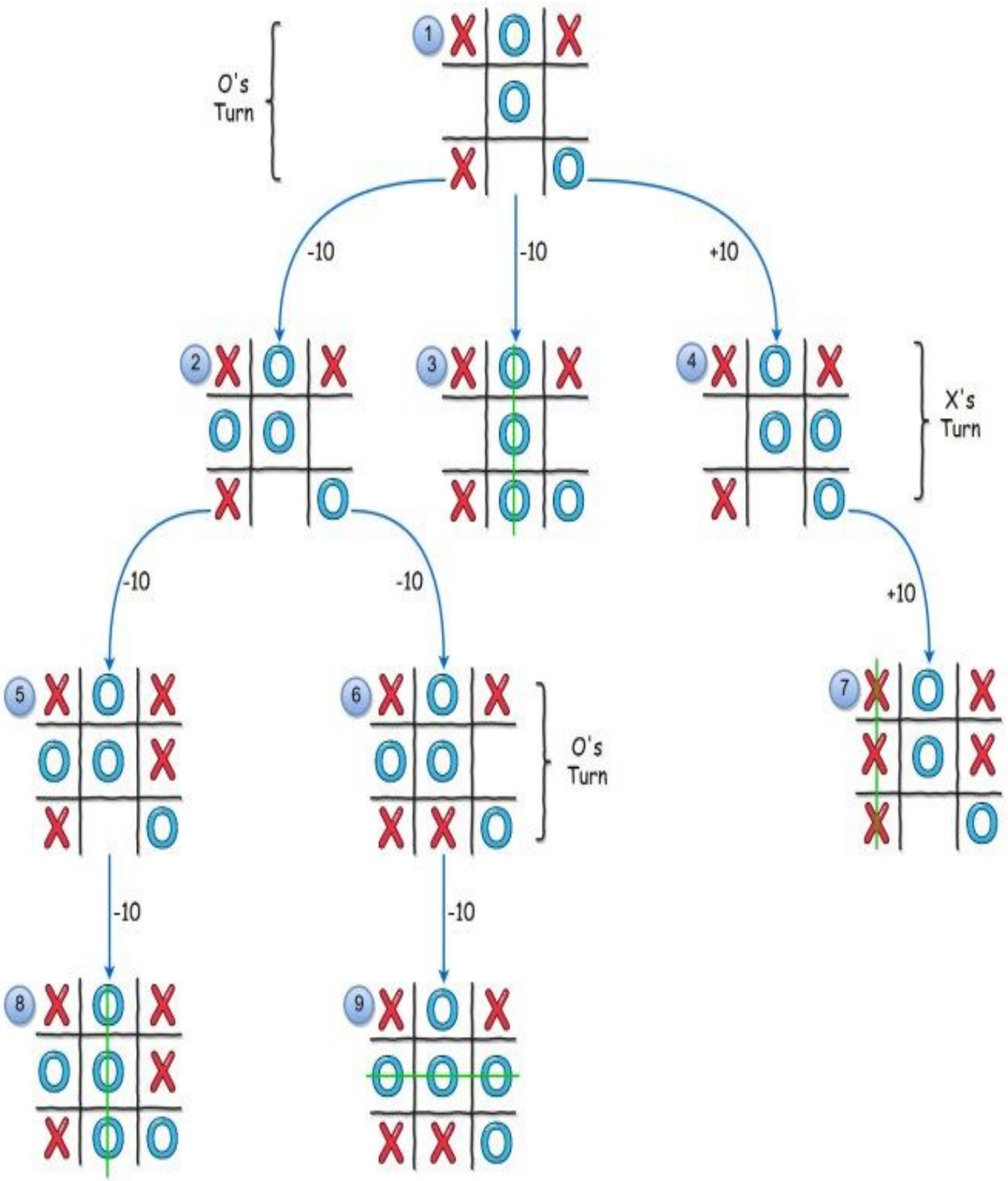
So if you are in state 1 (from the above diagram), you have 3 possible moves which lead to state 2, 3 and 4. For these moves the scores are –

- +10 if you choose state 2.
- 0 if you choose state 3 because it will be a draw if Player O is playing optimally.
- -10 if you choose state 3.

So conceptually we know that player X must choose the winning move. This is done programmatically by choosing the move which will return the maximum score. So, X will always try to maximize the score and will always choose that move which will fetch X the maximum score. Thus, in the above scenario X chooses the move which goes to state 2.

Now, to make sure you understand both sides of this algorithm. Let us take the same state as above and let us say it is O's turn now and not X's. Can you draw a state diagram which will depict all the possible moves and the scores which will be returned by playing them? You can! Just pause here for a moment and try to draw the state diagram for the above game, assuming it is player O's move.

You should get a diagram like this –



Did you get a diagram like this? Good job if you did . Player O has 3 possible moves and the scores are –

- -10 if you choose to go to state 2.
- -10 if you choose to go to state 3.
- +10 if you choose to go to state 4.

Player O will always try to minimize the score, so player O must select a move which will either lead to state 2 or 3 to win.

Writing code for MiniMax algorithm –

Writing code for MiniMax algorithm is not very difficult, but you may not get it in the first try so I'll help you out. Firstly, have a clarity on the smaller pieces of logic and write methods for them first. You will need these 3 helper methods for your code –

1. printGame(game) – which prints the state of Tic-Tac-Toe game.
2. hasPlayerWon(game, player) – which tells if the given player has won the given Tic-Tac-Toe game.
3. score(game) – which returns +10 if player X has won, -10 if player Y has won, 0 otherwise.

So now you have a little clarity over the smaller pieces, code them first. Now, you are ready to write the MiniMax algorithm method. It is a recursive method which takes in 2 inputs –

- the state of the game
- information on which player's turn it is.

It will need to return –

- max/min score which can be achieved for the given player for the given game state.

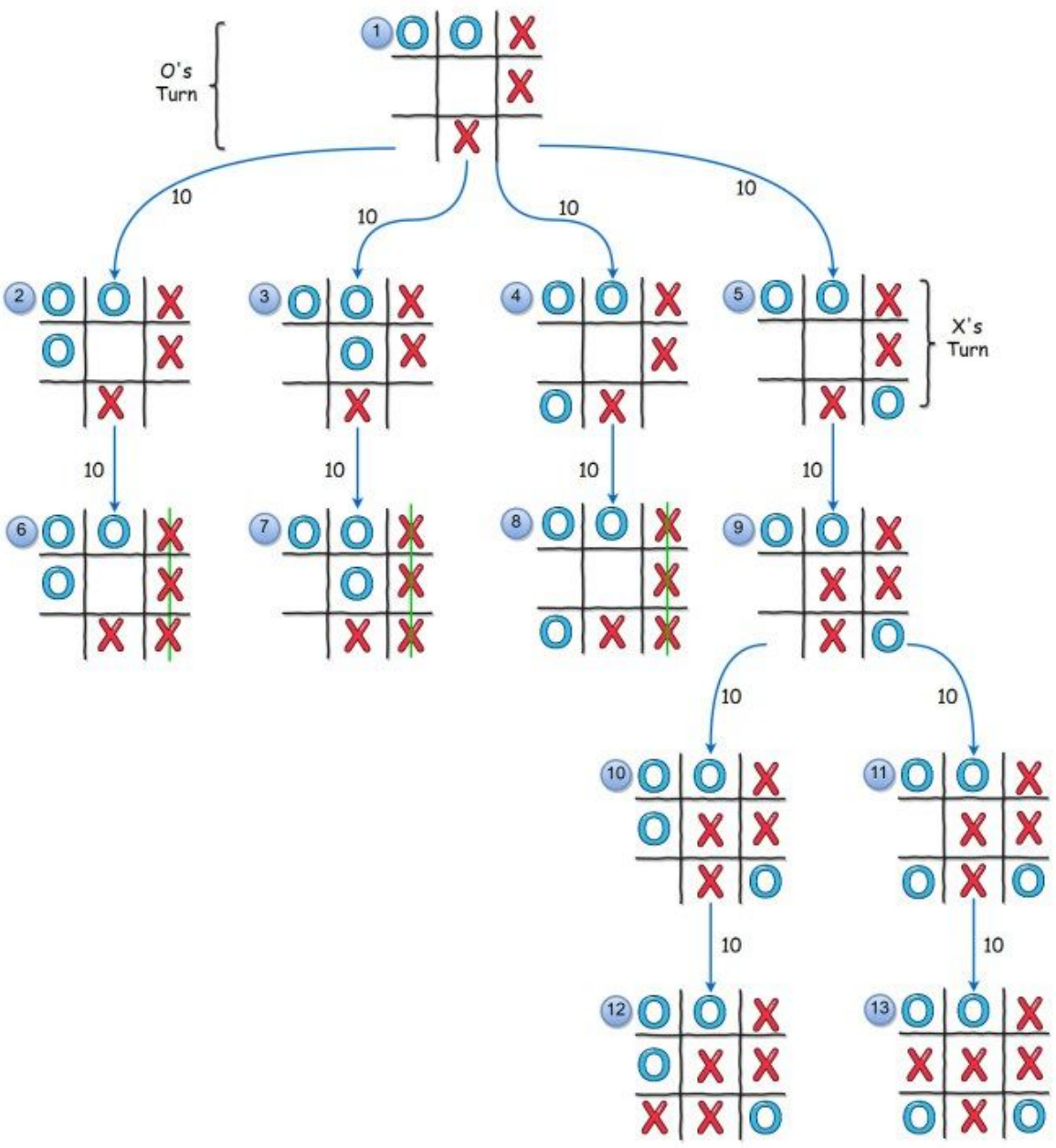
- best move which can be played by given player.

We can make a new class to return all the information we need. So, our MiniMax algorithm will look like –With your new clarity over the helper methods and the pseudocode, try to write the code for MiniMax algorithm. When in doubt come back and read the MiniMax algorithm theory and the implementation tips. Remember, you are trying to write the code for an ideal Tic-Tac-Toe player here, so you need to write the starter code for simulating a 2-player Tic-Tac-Toe game.

Refer minimaxwithoutdepth.txt for pseudo code!!

Ideal player doesn't give up!

There's just one thing lacking in our algorithm now. Take the case given below –

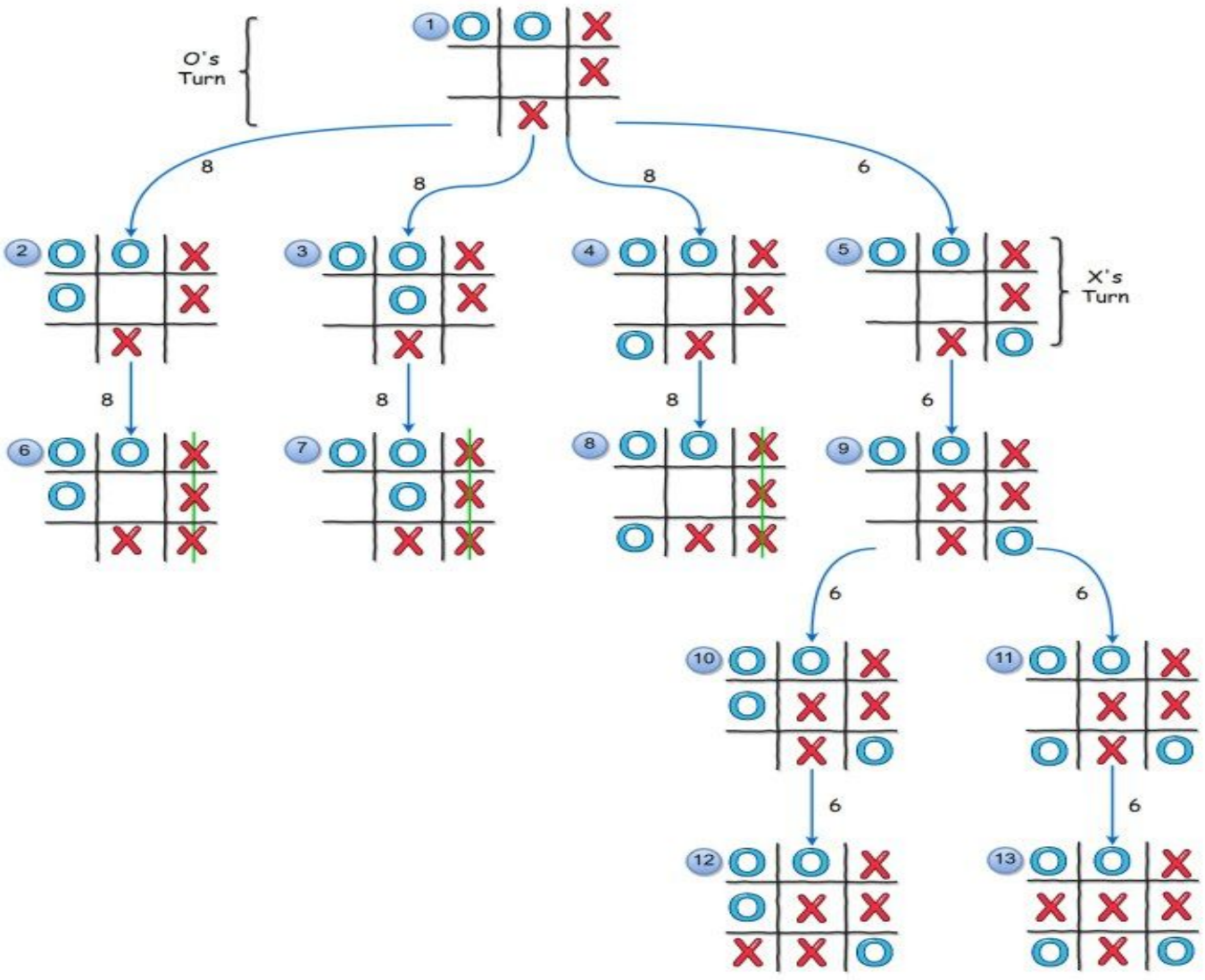


So for the above case, player O will lose no matter the decision taken. If you think about this, if we apply the Minimax algorithm we formed so far, in this case Player O would choose state 2. This is because state 2 is the first state it programmatically encounters while computing the minimum value.

But this doesn't seem right. Ideally, we would want Player O to go for state 5 because that's what an ideal player would do. An ideal player would choose that move in which he/she would loose the game in 3/4 turns rather than just the next turn.

How do we implement this in our algorithm? We can add another parameter "*depth*" to our algorithm and decay the score by the factor of depth. So, if a player wins by taking more turns, the gain would be lesser and if the player took lesser turns, the gain would be more.

If we apply this "decay score by level" logic to the above example, the case would look like this –



Now our player O will obviously choose state 5. Now our TicTacToe bot is an ideal bot! The pseudo-code now would look something like this –

Refer minimaxwithdepth.txt for pseudo code