

M,n,k-games solver

Algorithms and Data Structures - project 2022

Task description

Task is composed of two parts:

1. The first one is to implement an engine of a generalized version of tic-tac-toe.
2. The second one is to implement a program solving the game using min-max algorithm (<https://en.wikipedia.org/wiki/Minimax>).

NMK games family

The generalized version of tic-tac-toe, i.e. the family of games NMK (<https://en.wikipedia.org/wiki/M,n,k-game>), accepts 3 parameters: N, M, and K, where:

1. $(N \times M)$ is the size of a board,
2. The value (K) determines the victory condition. That is, K neighboring fields forming a continuous horizontal or vertical or diagonal line section give a victory.

Game engine

The engine of the game should allow to:

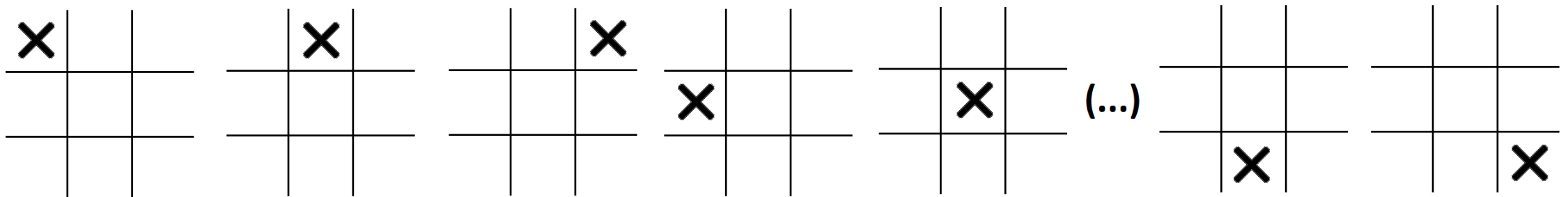
1. Generate all the moves.
2. Mark the state of the game. That is, it should produce an answer if the game has ended and the player won/tied/lost?

Moves generator

It is required to implement a deterministic generator of moves.

It's adds a token of a player (representing the move of the player, for example a cross, a circle, or white or black token) in deterministic manner.

For example, starting in left upper corner, first passing all the columns and then proceeding to the next row.



Moves generator

In the assignment the tokens of the first player (Player 1) are marked as '1', the tokens of the second player (Player 2) as '2', and empty fields as '0'.

Hence in the basic version of tic-tac-toe (NMK 3,3,3) the generated moves for an empty initial board are (keep in mind that the order is important):

```
100 010 001 000 000 000 000 000 000
000 000 000 100 010 001 000 000 000
000 000 000 000 000 000 100 010 001
```

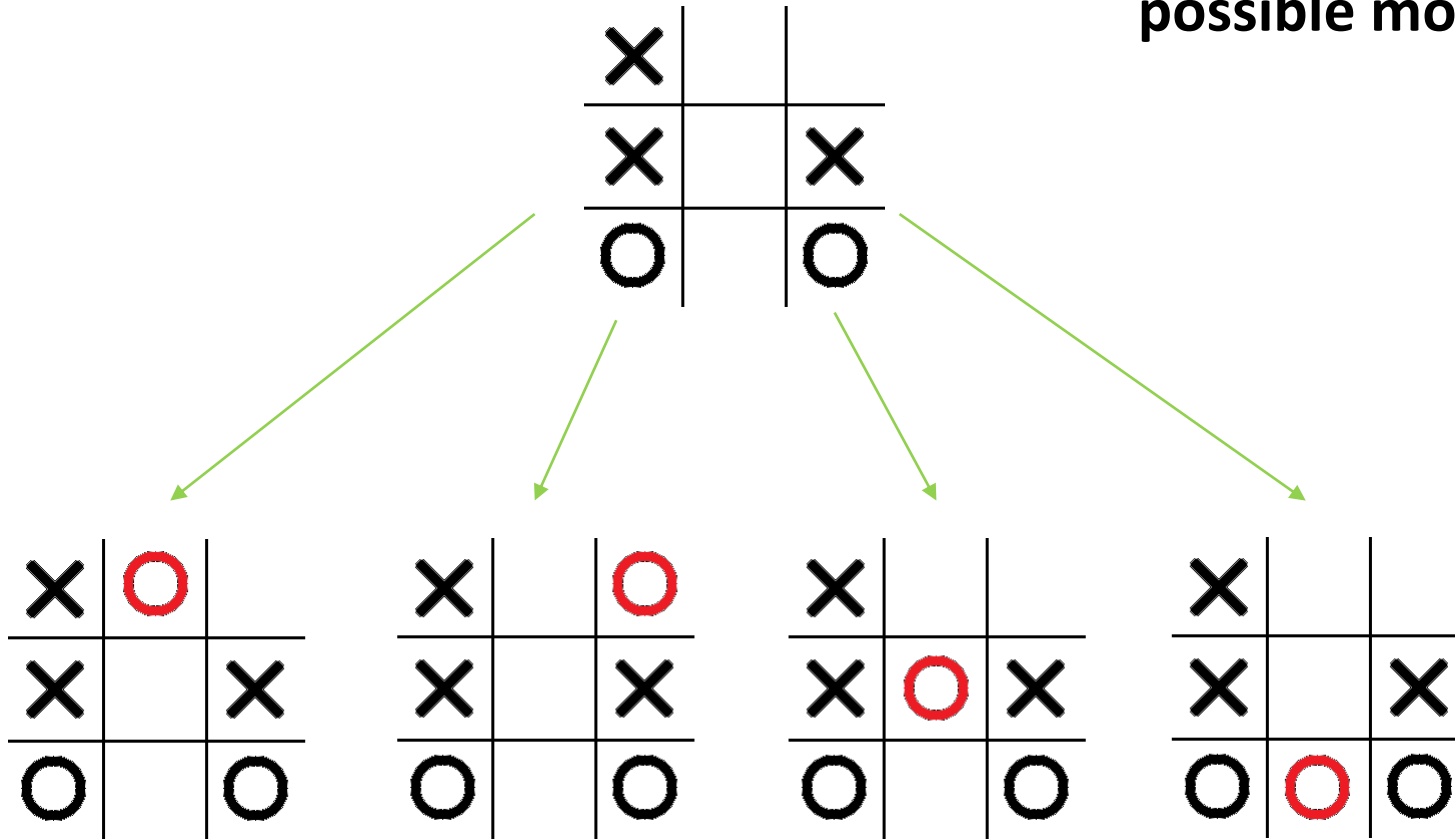
Moves generator

X		
X		X
O		O

possible moves count = ?

Moves generator

possible moves count = 4

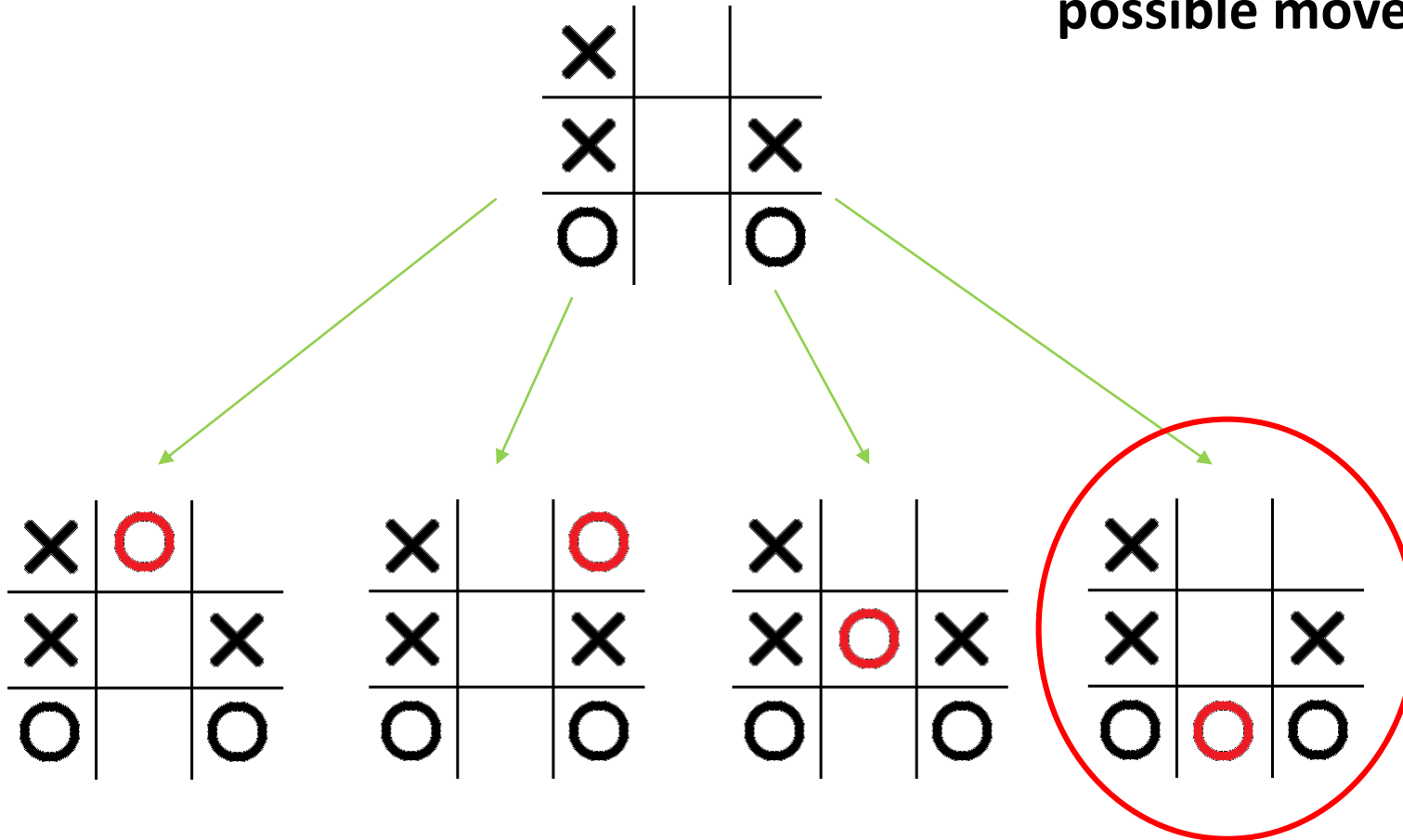


Moves generator

1. In the enhanced version, if one of the generated states is final (one of the players won or the board is full [which means tie]), then the engine should generate only one state.
2. If there are a few states possible, then it should generate only the first win/tie state.
3. Observe that if the newly generated state is win, then it can be only the win state for the active player; it is not possible, that the active player induces the winning of the opponent by her move.

Moves generator

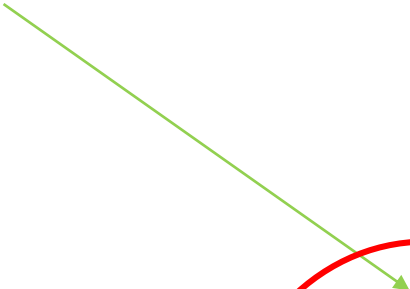
possible moves count = 4



Moves generator

X		
X		X
O		O

possible moves count = **1**



X		
X		X
O	O	O

NMK Solver

1. In the second part of the assignment it is required to implement an algorithm solving some, not necessarily initial, state of the NMK game.
2. One can use Minmax or Negmax with some enhancements.
3. Observe that in the case of this family of games the solution is always determined, because it is 2-players game, without randomness, and without hidden information.
4. Hence it is possible to determine if the game (if both players play optimally) ends in winning (losing) of a player or in a tie.

Minmax algorithm (pseudocode)

```
int Minmax(G gameState, Player activePlayer)
```

```
    int score = gameState.Evaluate(activePlayer)
```

```
    if (gameState.State == GameOver) { return score }
```

```
    allPossibleMoves = gameState.GeneratePossibleMoves(activePlayer)
```

```
    if (activePlayer == Player::first)
```

```
        forall (possibleMove in allPossibleMoves)
```

```
            best = maximum(best, Minmax(possibleMove, activePlayer.GetOponent()))
```

```
        return best
```

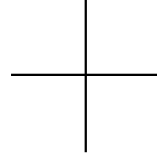
```
    else
```

```
        forall (possibleMove in allPossibleMoves) {
```

```
            best = minimum(best, Minmax(possibleMove, activePlayer.GetOponent()))
```

```
        return best
```

Minmax algorithm (n=2, m=2, k=2)



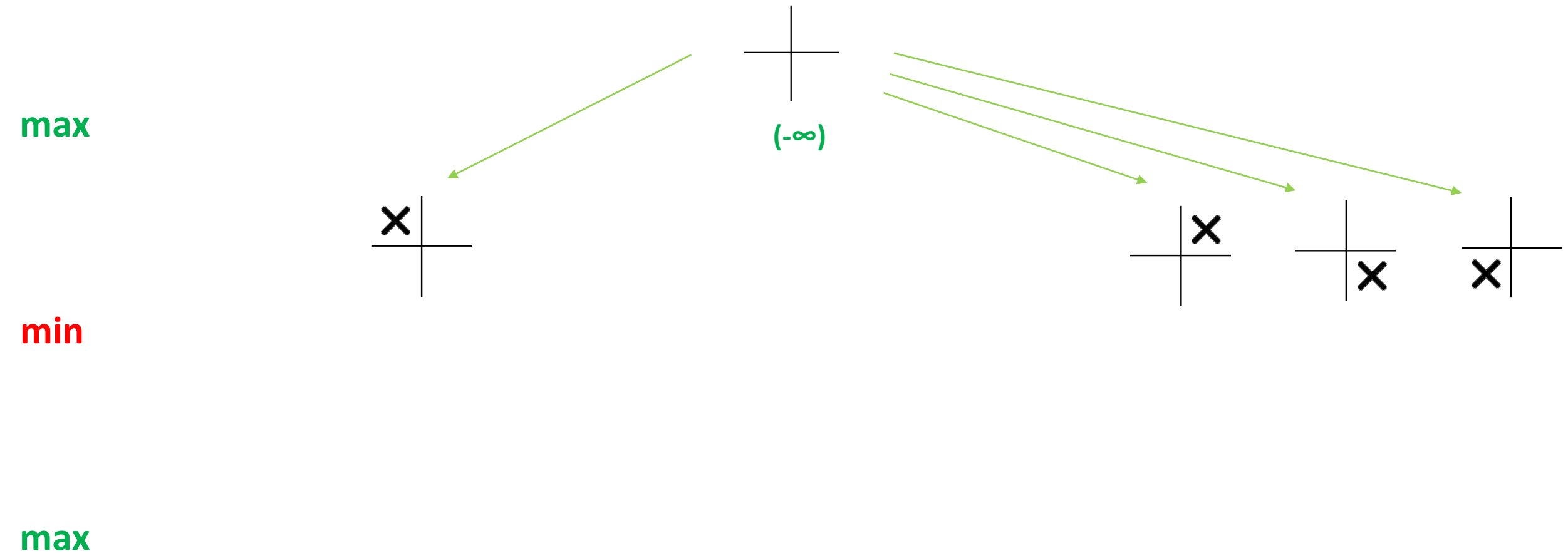
$(-\infty)$

max

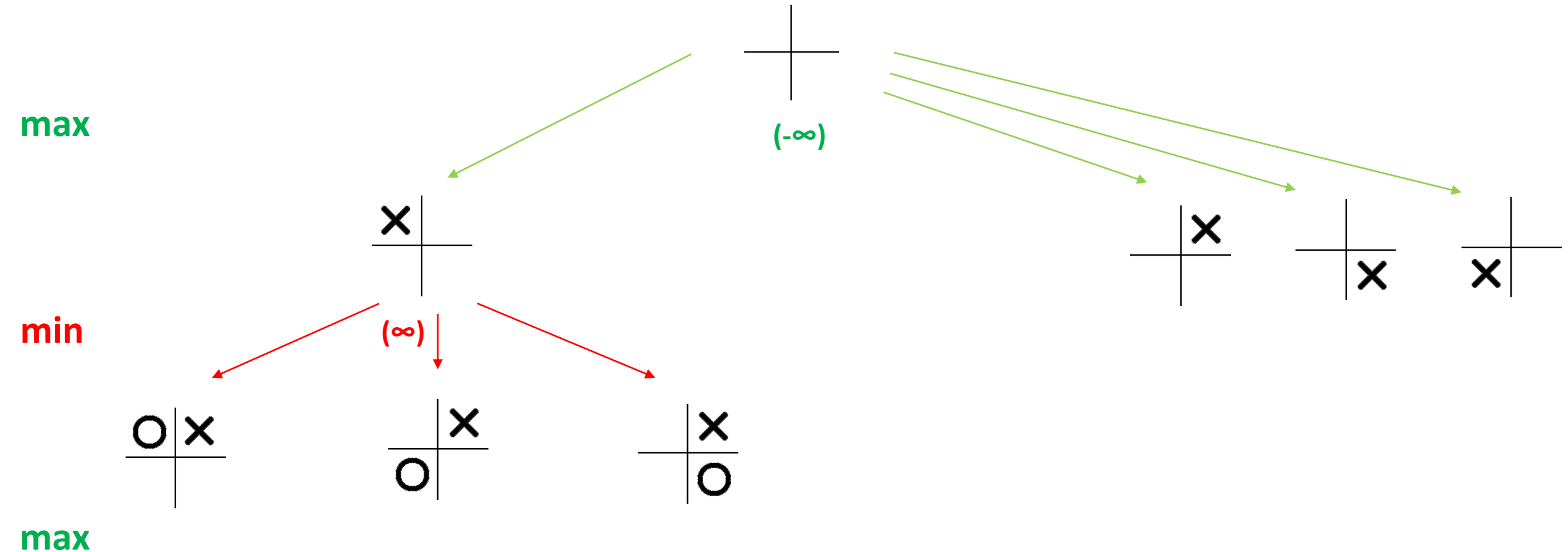
min

max

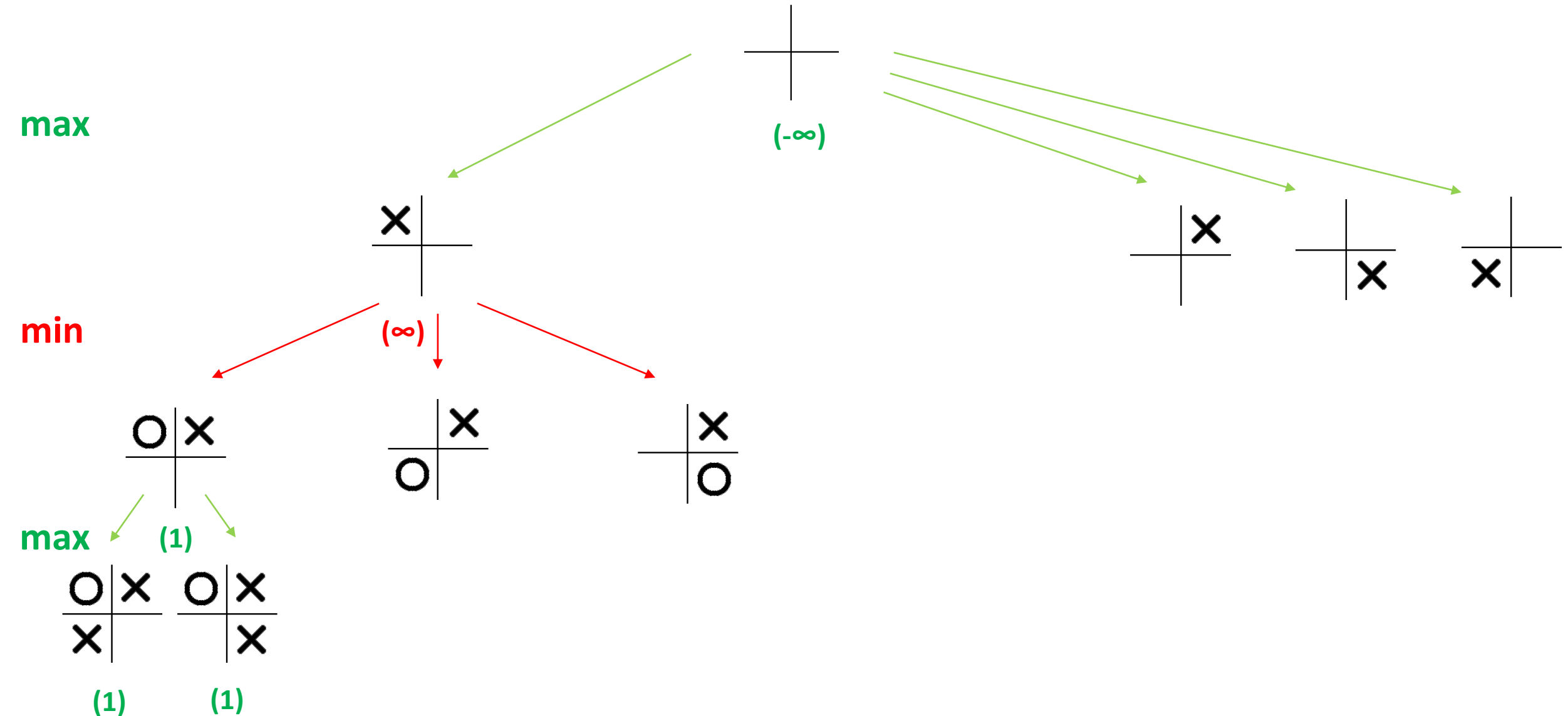
Minmax algorithm (n=2, m=2, k=2)



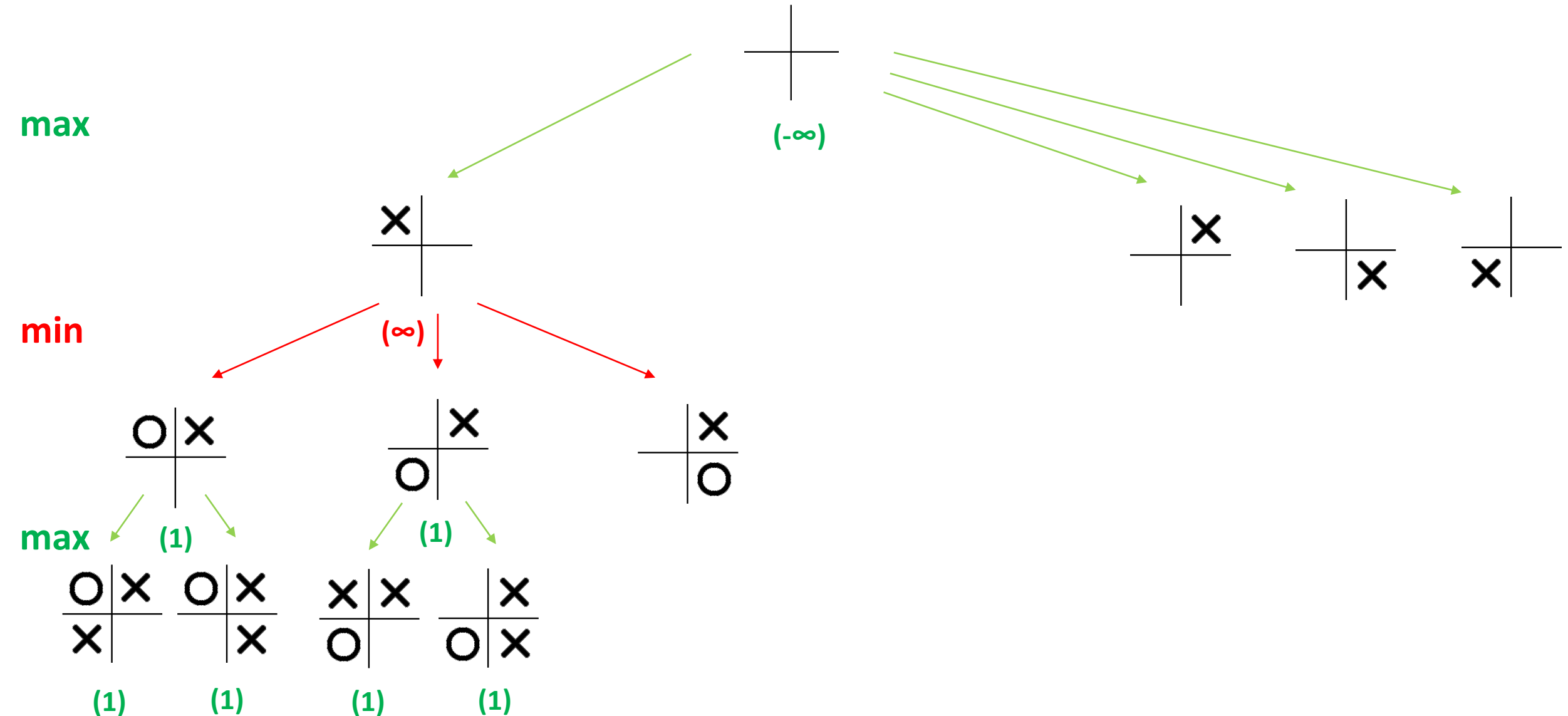
Minmax algorithm (n=2, m=2, k=2)



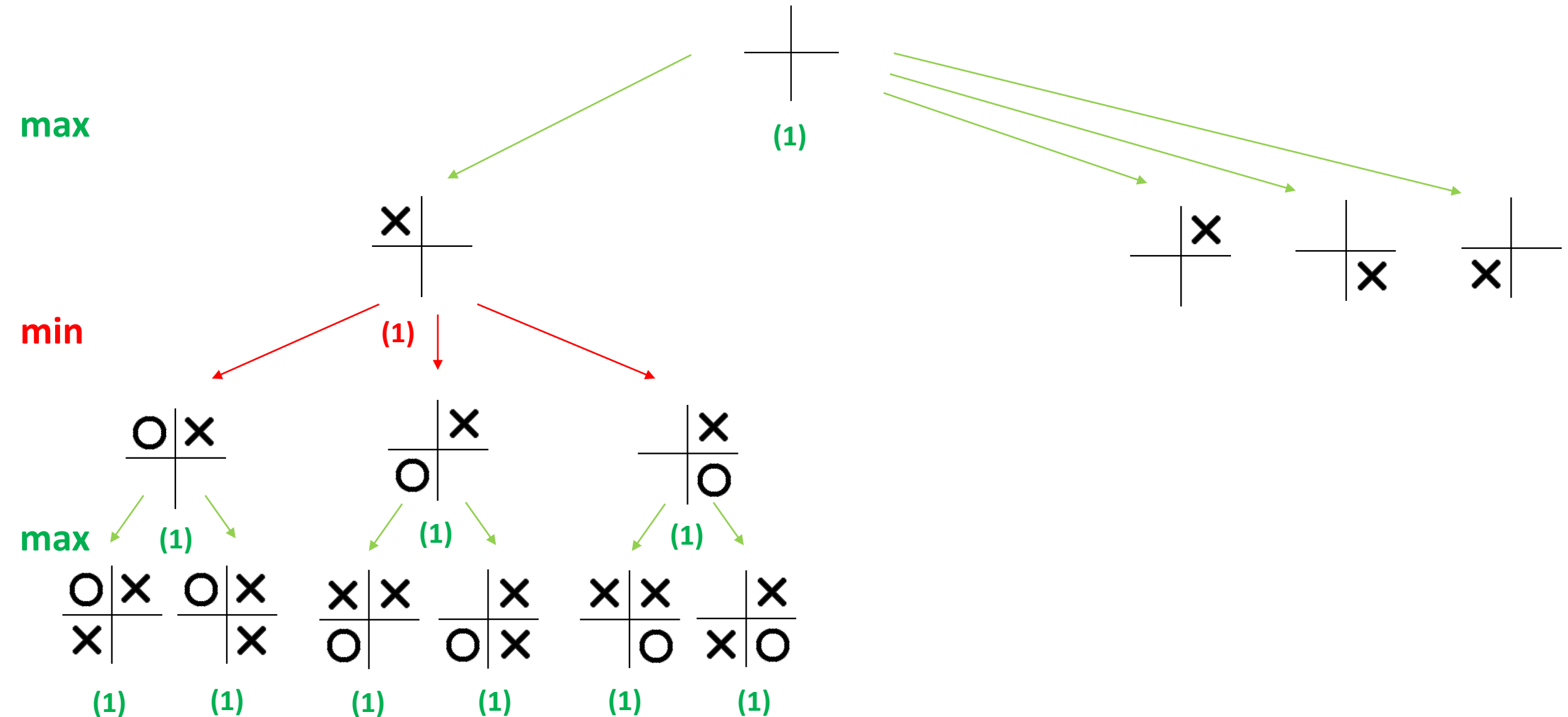
Minmax algorithm (n=2, m=2, k=2)



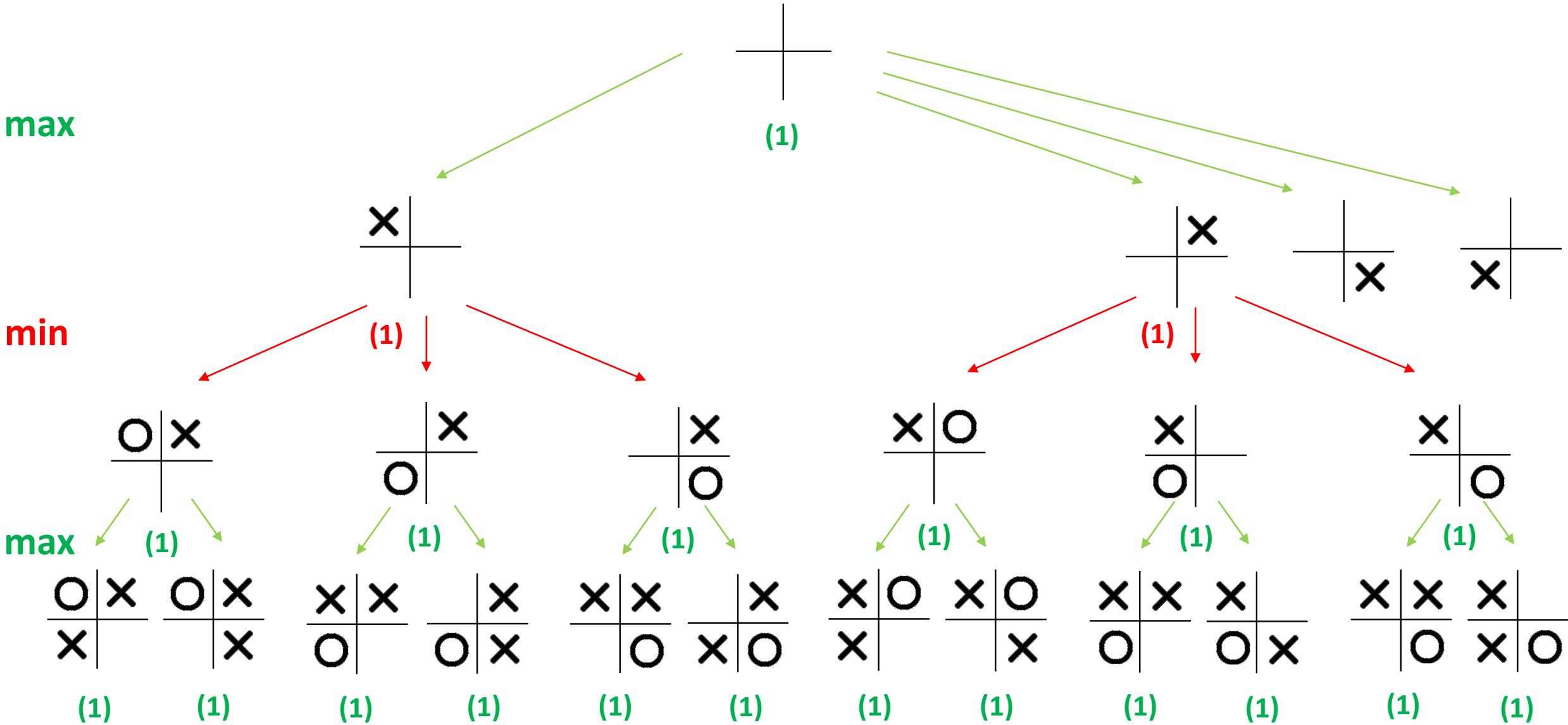
Minmax algorithm (n=2, m=2, k=2)



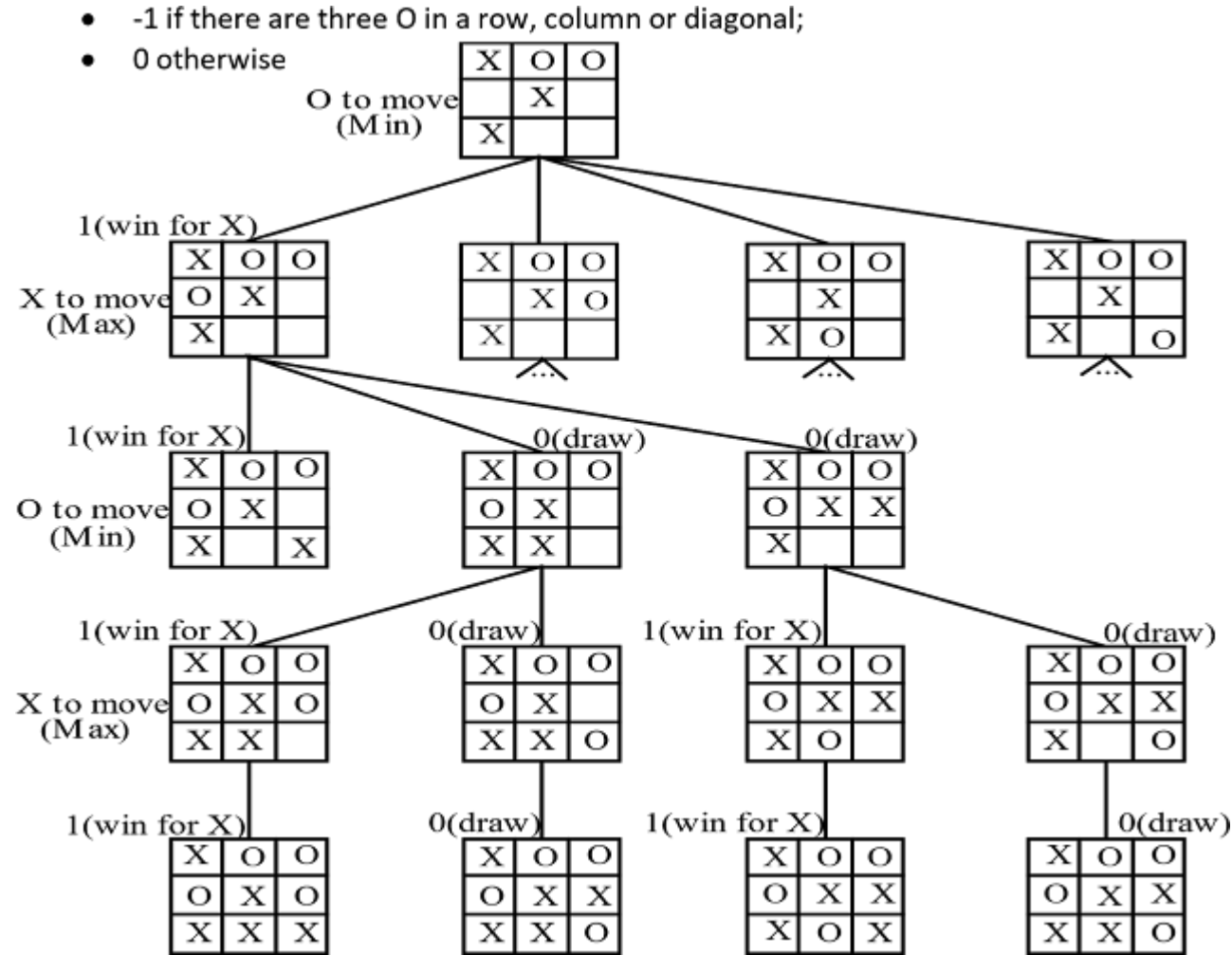
Minmax algorithm (n=2, m=2, k=2)



Minmax algorithm (n=2, m=2, k=2)



Minimax algorithm (Tic-tac-toe, $n=3$, $m=3$, $k=3$)



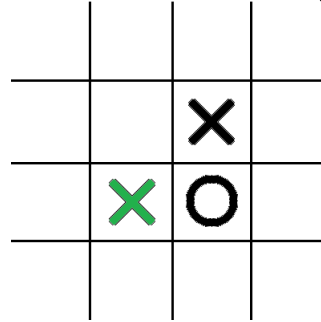
source: <https://csit-notes.blogspot.com/2019/08/minimax-algorithm-in-game-theory-and-c.html>

NMK Solver

1. The first enhancement for Minmax is to end the min search in the case of finding -1 or max search in the case of finding 1, because the results cannot be enhanced.
2. The second enhancement is the observation that there are situations that a player has winning move in the next move (provided that the situation did not appeared earlier for the opponent). Precisely it is possible that there is a sequence of $(K-1)$ tokens which can be completed on both ends.

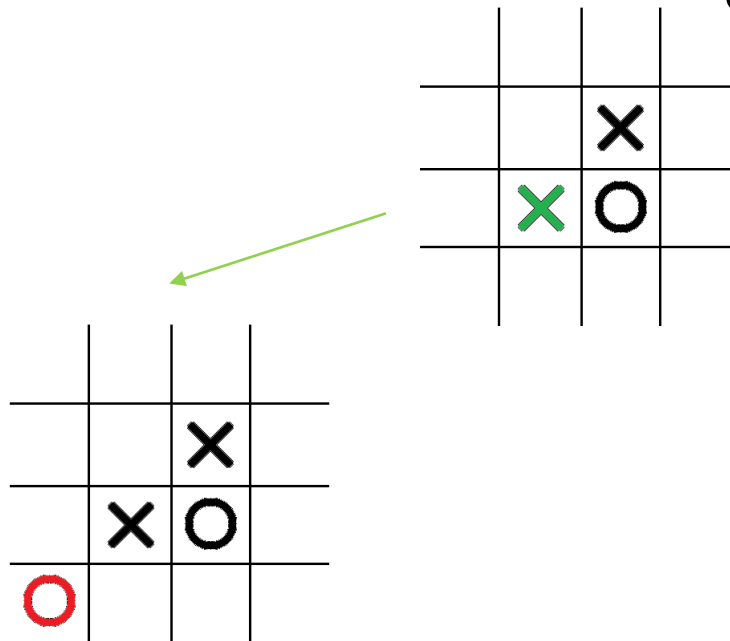
Threats

**The cross player wins
on his next move**



Threats

The cross player wins
on his next move



Threats

The cross player wins on his next move

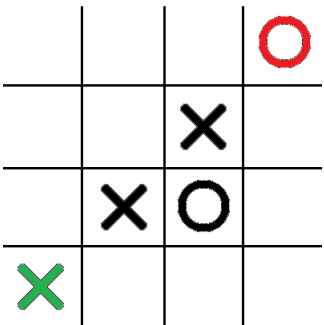
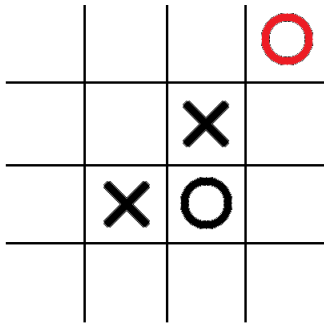
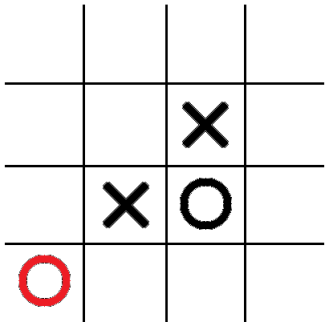
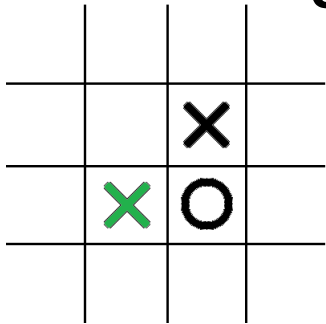
		X	
	X	O	

		X	
	X	O	
O			

			O
		X	
	X	O	
X			

Threats

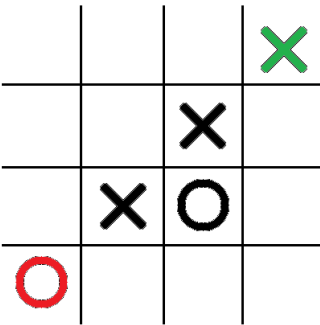
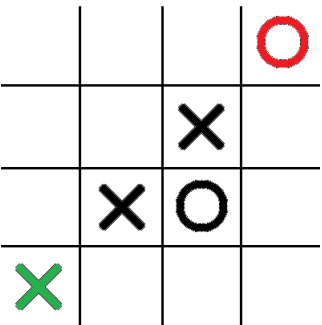
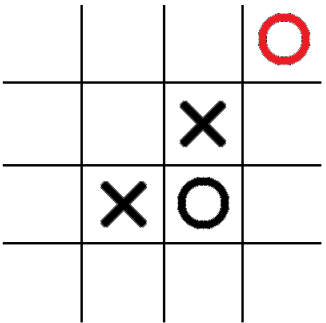
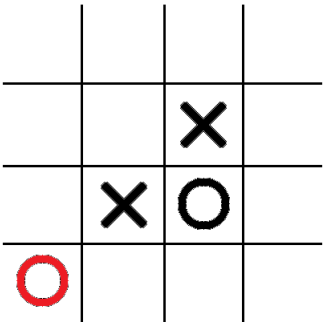
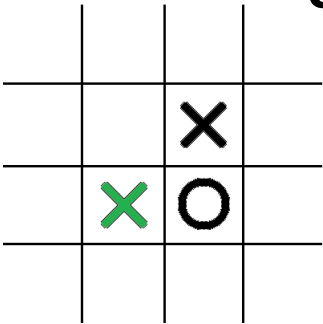
The cross player wins on his next move



Threats

The cross player wins
on his next move

Therefore, we can
mark this state as
terminal.



Test examples

1. GEN_ALL_POS_MOV N M K ActivePlayer - generate all possible moves and the number of moves.
2. GEN_ALL_POS_MOV_CUT_IF_GAME_OVER N M K ActivePlayer - generate all feasible moves and the number of such moves. If one of the moves results in win or tie, then generate only the first such a move.
3. SOLVE_GAME_STATE N M K ActivePlayer - solving the game and stating one of the possible answers:
 - FIRST_PLAYER_WINS
 - SECOND_PLAYER_WINS
 - BOTH_PLAYERS_TIE

Test examples (1)

Input:

	1 2 0	1 2 0
GEN_ALL_POS_MOV 3 3 3 1	1 0 0	0 0 0
1 2 0	0 0 0	1 0 0
0 0 0		
0 0 0		

Output:

	1 2 0	1 2 0
	0 1 0	0 0 0
	0 0 0	0 1 0
7		
1 2 1	1 2 0	1 2 0
0 0 0	0 0 1	0 0 0
0 0 0	0 0 0	0 0 1

Test examples (2)

Input:

GEN_ALL_POS_MOV_CUT_IF_GAME_OVER 3 3 3 1

0 2 1

2 2 1

0 1 0

Output:

1

0 2 1

2 2 1

0 1 1

Test examples (3a)

Input:

SOLVE_GAME_STATE 3 3 3 2

1 0 0

0 0 0

0 0 0

Output:

BOTH_PLAYERS_TIE

Test examples (3b)

Input:

SOLVE_GAME_STATE 3 3 3 1

1 2 0

0 0 0

0 0 0

Output:

FIRST_PLAYER_WINS

Test examples

1. All the tests used in the assignment are available at e-nauczanie (e-learning) platform.