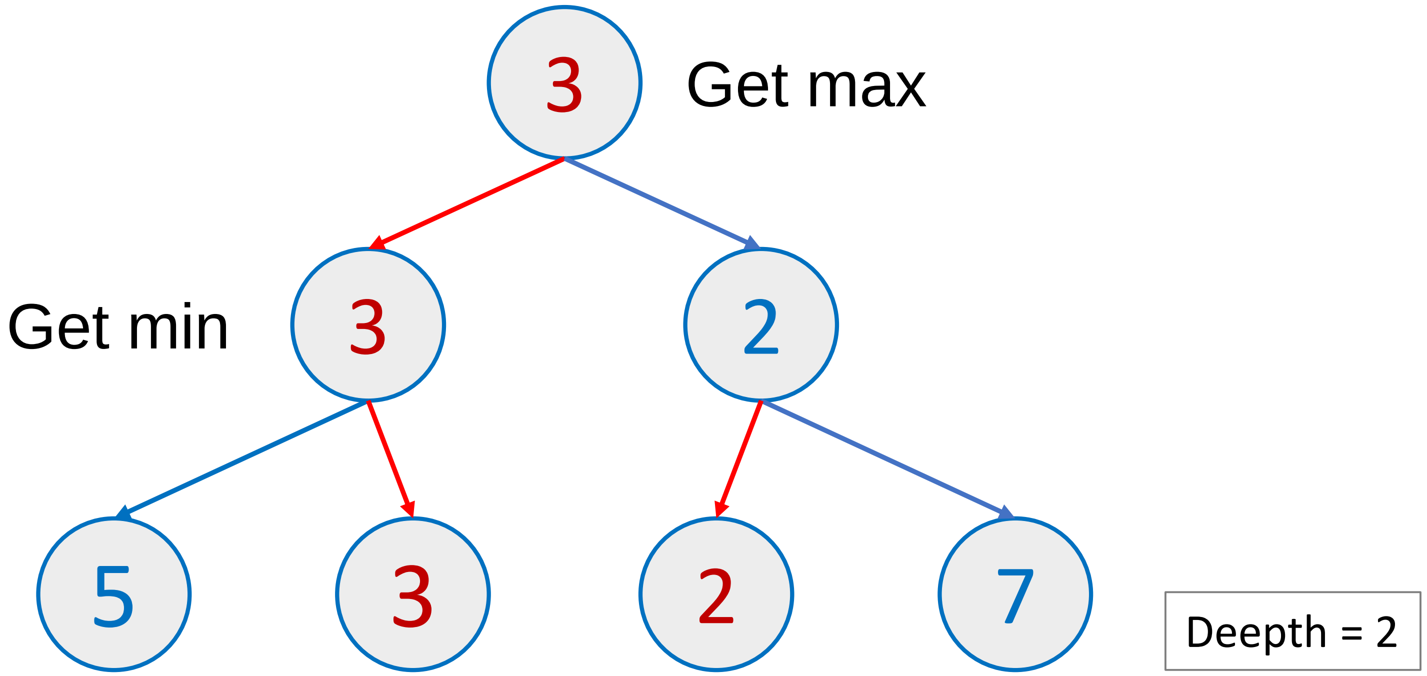
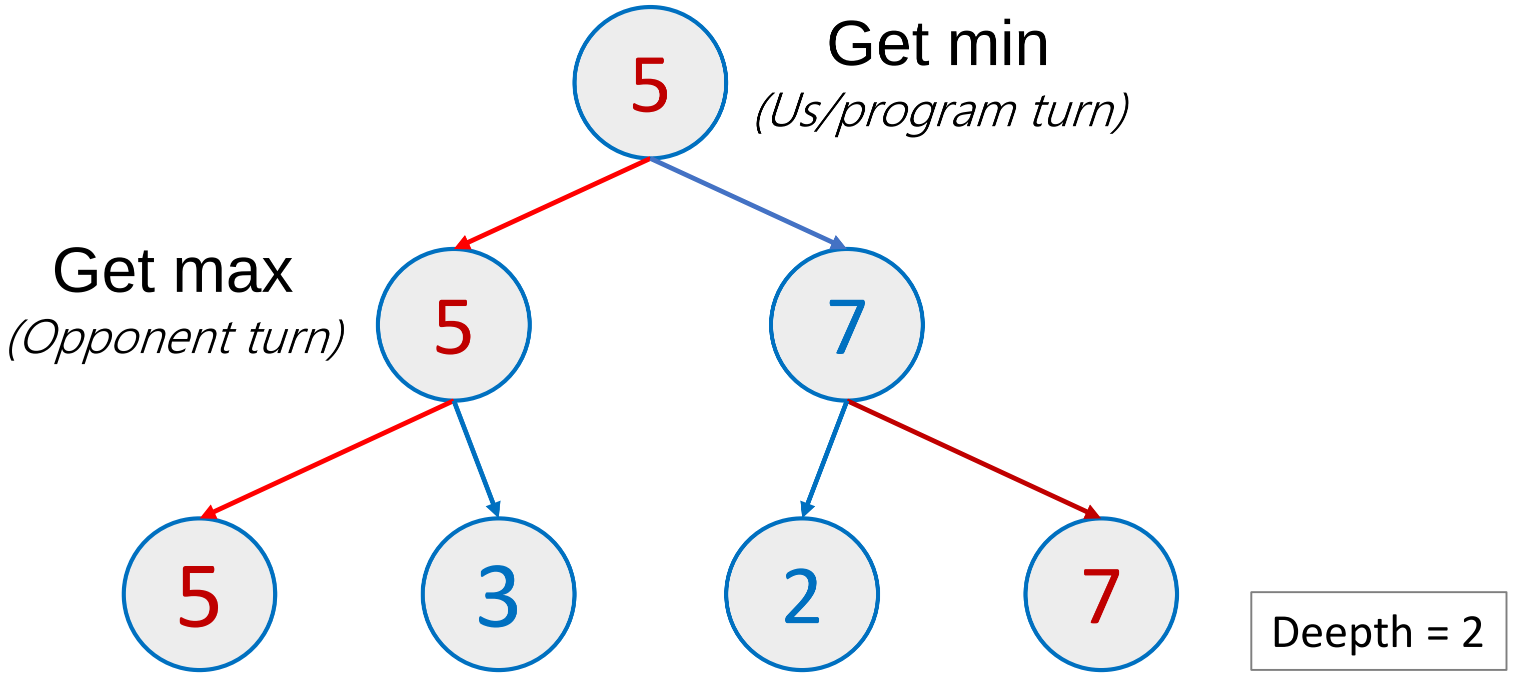
MiniMax Algorithm

- MiniMax algorithm help us/program make a decision which is the best move in all of state can happen.
- To basically, the whole algorithm is divided into two-phase. Get max and get min from children of each state.



MiniMax Algorithm

- When we combine this algorithm with Adversarial Search, we must predict our's opponent move. Suppose they always select the best move for them. So, we change our algorithm a little bit.



MiniMax Algorithm

Pseudocode

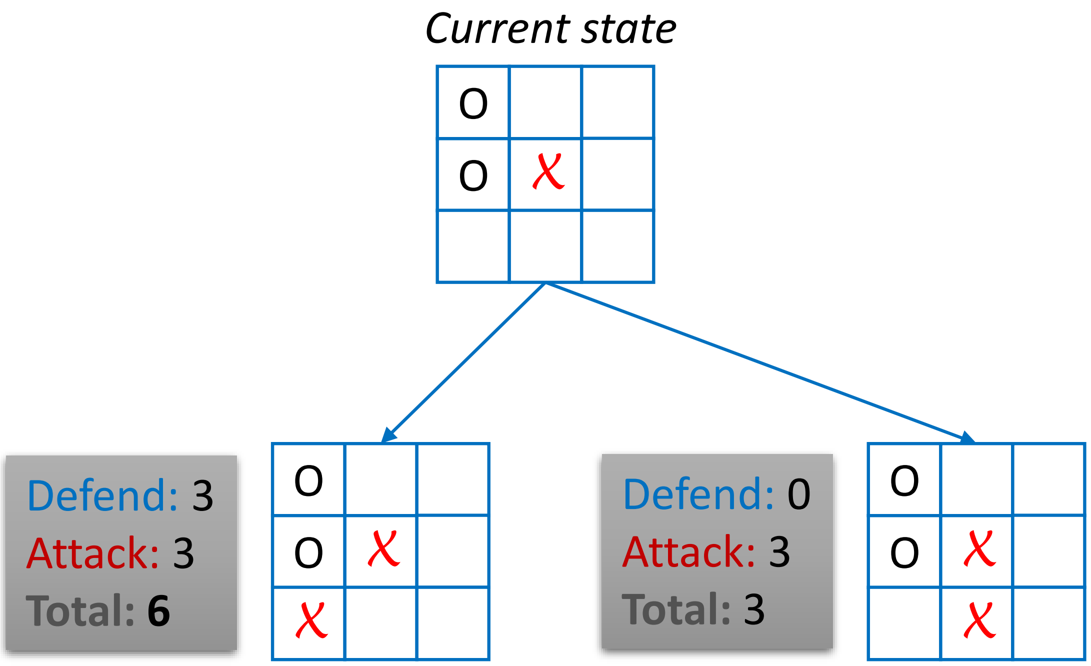
```
def mini_max(crr_state, deepth, turn):
    if deepth == 1:
        return get_max_benefit(crr_state)
    else:
        best_move = None
        crr_benefit = 1000#if deepth is odd, 0 if not
        for move in can_move(crr_state, turn):
            get_benefit = mini_max(move, deepth - 1,
                                             change_turn)
            if get_benefit > crr_benefit and
             deepth%2==1 or get_benefit < crr_benefit
             and deepth%2==0:
                crr_benefit = get_benefit[1]
                best_move = move
        return best_move, crr_benefit
```

Profit/Benefit Function

The benefit of a state depends on two main part:

- How many point can earn from defend move.
- How many point can earn from attack move.

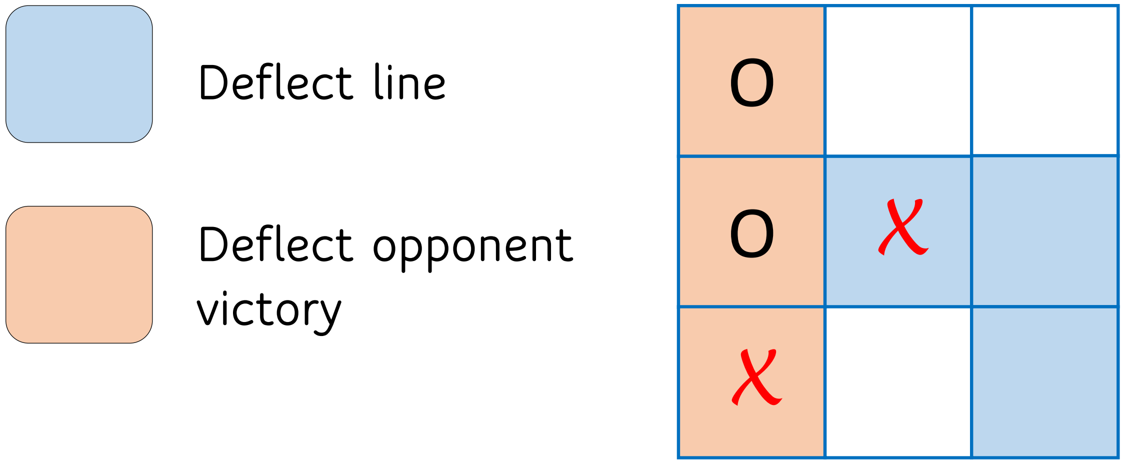
  ** Do not do stupid move.*



Profit/Benefit Function

Point of defend move in caro/gomoku game consist of:

- How many lines you can deflect, prioritize the longest line first.
- Move into the position that our opponent can get the victory.

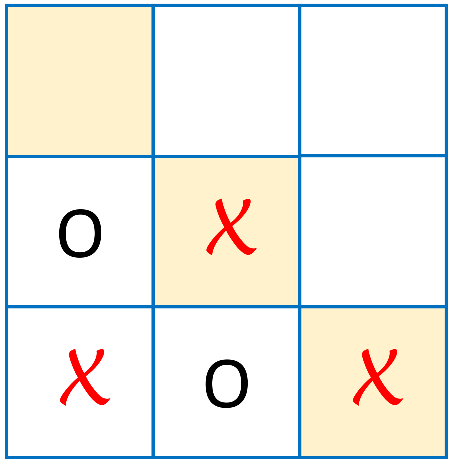


Profit/Benefit Function

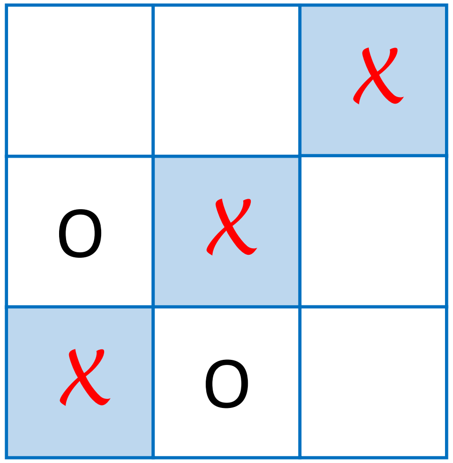
Point of attack move in caro/gomoku game consist of:

- How many lines you can make, prioritize the longest line first.
- Move into the position that we can get the victory.

Make a line



Get the victory

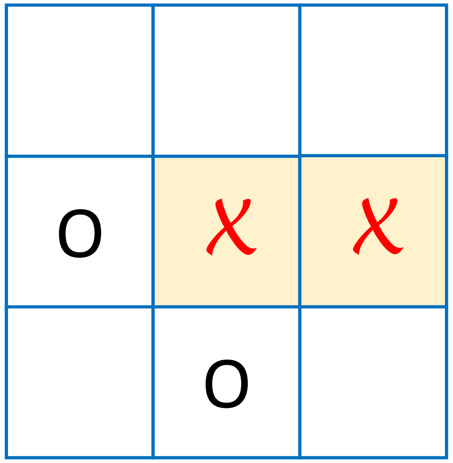


Profit/Benefit Function

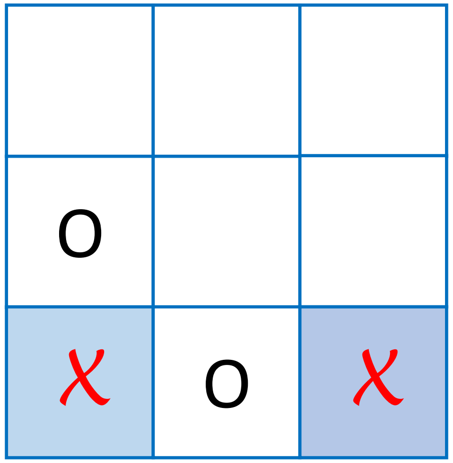
Stupid move is:

- Move to the position can't get the victory after that. (1)
- Move into the position not necessary to defend. (2)

(1)



(2)



Profit/Benefit Function

Pseudocode

```
def get_max_benefit(crr_state):
    best_move = None
    crr_benefit = 0
    for move in can_move(crr_state, turn):
        crr_pos = get_crr_move(move, crr_state)
        get_benefit = depend(crr_pos) + attack(crr_pos)
        if get_benefit > crr_benefit and
        not is_stupid_move(crr_pos):
            crr_benefit = get_benefit
            best_move = move
    return best_move, crr_benefit
```