※이 레퍼런스는 state가 env. 그 자체라서 이 부분을 수정해야 함.

class Node : input : ~~move~~ (action), state, parent, (player) — (env. → state)

    action (env. <state)
    child_node_list.
    farther_node
    visit_cnt
    win_cnt
    available_actions
    player

    def select_child :
        UCT 공식에 따라 child node 선택.
        return : UCT값이 최대인 child node.

    def add_child : input : action, state.
        expansion에 해당.
        n = Node(action = action, parent = self, state = state (deepcopy))
        available_actions 에서 action 제거.
        child_node_list 에 state, action 에 따른 node 추가.
        return : n. (action, state를 갖는 childnode)

    def update : input : result (win/draw/lose)
        이 노드 업데이트. Backpropagation에 해당.
        visit_cnt += 1 # 한번 거쳤으니 visit +1
        win_cnt += result

def UCT : input : rootstate (env.), itermax (player)
    rootnode = Node(state (env) = rootstate) (player.)

    for i in range(itermax): (→ 한 번 수를 두기 전 MCT 알고리즘을 얼마나 실행?)
        node = rootnode.
        state = rootstate (deepcopy)

        # selection
        while node 에서 가능한 action이 없음.
            & node 의 childnode가 1개 이상 존재 : (가능한 action이 없고 childnode가 없음. ⇒ terminal node.)

            node = node.select_child() # UCB1 알고리즘에 따라 선택한 node.
            state → node.action 로 바뀐 state. # 이걸로 한 수 진행.

        # expansion # selection 후 아직 플레이되지 않은 action을 하나 선택해서 진행.
        if node 에서 가능한 action 존재. :
            action = node 에서 가능한 action 中 랜덤으로 1개 추출
            state → action 로 바뀐 state.
            node = node.add_child(action, state)

        # Simulation (state이 게임종료가 아님.) # selection → expansion … 이 child node 기준으로 simulation.
        while ~~state 에서 가능한 action이 존재~~ :
            state → 가능한 action 하나 뽑아서 행동. → 바뀐 state.

        # Backpropagation
        while node 가 최상위 node가 아님. :
            node.update(state 에 따른 result)

    여기까지 하면 root node의 childnode들이 싹 업데이트됨.
    s = sorted(rootnode.child node_list, key = lambda c : c.wins/c.visits)

    return : sorted(s, key = lambda c : c.visit_cnt)[-1].action
    그냥 s[-1].move 해도 되지 않을까?

※ sorted()는 낮은 수부터 정렬.
⇒ [-1] 값이 최댓값.

? 여기서 sorted 한게 의미없지 않나? →

이렇게 하면? →