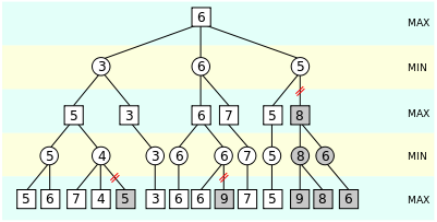
CANTRIS

DESIGN PHILOSOPHY

- Search strategy
  使用 minimax with alpha-beta pruning 當作我的搜尋策略



  - board 1、3
    6*3: game tree depth=max(row, col)
    8*4: game tree depth=5

```python
def _minmax(game, board, player, opp, curr, alpha, beta, depth, mypts, opppts):
    if game.checkgameover() or depth <= 0:
        score = mypts - opppts
        return (score, None)

    moves = game.possible_moves(board=board)  # return all possible moves
    move_to_return = None
    for move in moves:
        ns, sim_mypts, sim_opppts = game.next_state(board, move, mypts, opppts, player, curr)

        score, position = _minmax(game=game, board=ns, player=player, opp=opp, curr=opp if curr == player else player,
                                  alpha=alpha, beta=beta, depth=depth - 1, mypts=sim_mypts, opppts=sim_opppts)

        # alpha beta pruning
        if curr == player:
            if score > alpha:
                alpha = score
                move_to_return = move
            if alpha >= beta:
                break
        else:
            if score < beta:
                beta = score
            if alpha >= beta:
                break
    # max level or min level
    if curr == player:
        return (alpha, move_to_return)
    else:
        return (beta, None)
```

```python
def make_decision(self):
    if self.turn == 1:
        opp = 0
    else:
        opp = 1
    board = copy.deepcopy(self.board)
    score, point = _minmax(game=self, board=board, player=self.turn, opp=opp, curr=self.turn, alpha=-float("inf"),
                           beta=float("inf"), depth=self.depth, mypts=copy.deepcopy(self.mypoints),
                           opppts=copy.deepcopy(self.oppopoints))
    x = point[0]
    y = point[1]
    return [x, y]
```

  - board 2
    10*5: game tree depth=8、cnt=1
    跟 6*3 & 8*4 不一樣的是:在_minimax function 內利用 sim_cnt 計算 player 跟 opponent 回合的次數，sim_cnt==2(兩回合結束)的時候 curr 換下一位，否則維持原樣

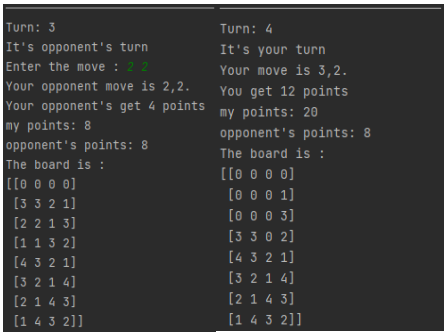
```python
for move in moves:
    ns, sim_mypts, sim_opppts, sim_steps = game.next_state(board, move, mypts, opppts, player, curr, steps)

    sim_cnt = copy.deepcopy(cnt)
    if sim_cnt == 2:
        curr = opp if curr == player else player
        sim_cnt = 1
    elif sim_cnt == 1 and sim_steps % 2 == 0:
        sim_cnt = 2
    score, position = _minmax(game=game, board=ns, player=player, opp=opp, curr=curr, alpha=alpha, beta=beta,
                              depth=depth-1, mypts=sim_mypts, opppts=sim_opppts, steps=sim_steps, cnt=sim_cnt)
```

DISCUSSION

- Motivation
  1. 相比 minimax，alpha-beta pruning 會省去一些計算的時間。
  2. 由於 CANTRIS 是 2-player 的棋盤型遊戲，盤面變化跟分數計算較簡單有原則，因此適合使用 minimax algorithm 當作 game tree 的 search strategy。

- Challenge & Improvement
  1. 在 10*5 跟 8*4 的 board，由於 game tree 較大，depth 若取 max(row, col)時間會超過，因此分別取 5 跟 8 當作 depth。
  2. 由於跑 minmax 會預測未來的結果，因此若在 class AI 內 call self 時會導致改變做決定前的數值(ex: board、turn、mypts、oppts)。
     這個問題卡了一段時間，後來找到的解決辦法是將_minmax 改成 global 的 function，而非宣告在 class AI 內，並再複製一份 board 當作跑 game tree 的 board`board = copy.deepcopy(self.board)`
     (mypts、oppts 同理也是採用此種方法)

  實際遊戲操作畫面:
  有 player 連續 combo 的情況發生，同時 algorithm 也會避免造成對手有 combo 的情況



- What I learned from the project
  實作出 alpha-beta pruning 的部份讓我學習到蠻多，不只是 minimax、game tree 的概念還有使用 class 必須考慮到的問題，對於 game agent 也有充分的了解，不再只是書上演算法的理解。
  同時也慶幸自己有早點開始做這份作業，因為原本 project 只有一個，突然多出了兩個，若按照我原本的行事作風，會拖到最後快死了才在趕死線，其他兩個 project 一定會來不及，因為中間 project 1 就比預期的卡了還久。