CS404-Assignment3

Alper Kaan Odabaşoğlu

May 2023

1 Game Tree Search Model

1.1 Players

There is 2 player for the game as stated in the Assignment document. Since this is a two player zero sum game, one of the player is max which tries to maximize the utility function and other is the min which tries to minimize the utility function. In our situation, one of the player is the computer (AI) and the other player is the human. Depend on which player starts, it is assigned as max or min. For instance, if human player starts first, it is assigned as max (tries to maximize the utility) vice-versa for AI.

1.2 States

States are objects that includes circle information for grid intersects, grid itself and turn information which remarks max or min turn. The object is given below for better understanding:

```
class State():
    def __init__(self, point_intersect, intersect, grid, turn, point):
        self.grid = grid
        self.turn = turn
        self.point = point
        self.intersect = intersect
        self.point_intersect = deepcopy(point_intersect)
```

Point intersect here keeps the main circle points to be utilized while giving points after a move. It is immutable since we need to use main numbers in intersects while giving points. Whereas intersect itself is mutable and whenever a diagonal line enters to a circle in intersect we decrease its point by one. Whenever the point in a circle in intersect matrix become 0, we go and get the point from immutable point intersect and write it to our player.

1.3 Initial State

Initial state is basically composed of empty grid with no diagonal lines are placed. Turn is 'max' (tries to maximize the utility function) and point is 0. Intersect is equal to point intersect for initial state since we do not place any diagonal line yet which can change the content of any circle in intersect matrix of a state.

1.4 Terminal State

Terminal state is the state where all the grid cells are full-filled with diagonal lines and no empty grid cell remained. Turn for terminal state can change depend on whether the grid has even or odd number of cells. If it is even, turn for the terminal state is 'min'. In other words, if human starts, AI will finish and vice-versa in the grid has even number of cells and since the starter player initialized as 'max', the finisher will be min for even case. Vice-versa for odd case. Point intersect matrix for terminal state is same for the initial state. However we cannot say anything about intersect since it can change based on the positioning of diagonal lines.

1.5 State Transition Function

State transition function basically handles the transition between one state to another given a move on that state. It takes 2 parameters which are the current state and the move that will be applied on that state and returns the new state which is gathered by the implementation of the legal move (must place the diagonal line into empty cell) on current state. To explain better, the pseudo-code is given below:

Input:
state: current state
move: legal move that can be applied on the current state

```
def state_transition(state, move):          apply move on grid of the state
        if turn is max
                change turn to min
                update point of the state
        else
                change turn to max
                update point of the state
        new state = updated state
        return new_state
```

Output:
new_state: new state gathered by applying move on state

1.6 Pay-Off Function

Before the pay-off function, as we stated above, in each state we gather and update points. So while we are building the tree, we also keep the cumulative points of players up to any node's state. In the pay-off function we directly utilize these points. If the player is max we return negated point. Else, we directly return the main point gathered so far. After that using this pay-off function, the alpha-beta pruning algorithm is implemented. In the game, utilizing alpha-beta pruning and utility function (pay-off), we gather evaluation results for each child of current node and based on whether it is 'max' or 'min', we select the next node (next state to proceed).

```
pay_off(node):
        return -(node's point)
```

Since the mxm grid is even, if the turn with max starts the game, turn with min finalize the game in each case. So, in other words, last move is taken by the 'min' player. Since the pay-off function based on the terminal states, we need to consider the player turn which finalize the game (min). Due to this fact and implementation decisions, we need to return (-1 $\cdot$ nodes) point. With this utility function, the algorithm behaves correct. This mainly ensures the correctness due to the callbacks of alpha-beta pruning algorithm. Since the terminal always reached with min's turn, we need to negate the point and call the evaluate function based on this negated terminal point. Otherwise, the selection for evaluate function do not work properly in the implementation. Finally, the important point without considering the implementation, the utility function based on the node's cumulative point that is acquired with successive player moves.