Contract Bridge Card Game

-using minimax algorithm

Contents:

- *Rules*
- *Plan*
- *Algorithm Used*
- *Implementation of Algorithm*
- *Explanation of minimax function*

All About the Game:

1.Rules:

- *Contract Bridge car game is a card game to be played by group of four.*
- *The game is always started with the player having the card "Spade-7".*
- *When he first puts the card, the player next can only put the cards closest to "Spade-7". Here in this case, he would have to choose from ("Spade-6","Spade-8","Diamond-7","Heart-7","Clubs-7").*
- *This means, he could always choose a card which is closer to the card put by previous player so that it could be best for him.*
- *Finally, the player who empties all his cards first wins the game.*

2.Plan:

- *Out of the four players, three were to be played by the computer and, one is to be played by the user.*
- *User has list of all available cards for him, from which he's supposed to choose a card best for him.*
- *In case of computer, this is achieved by selecting a card which is closer to previous card.*
- *If there are two cards of same priority(or distance), then the one closest to the "Ace" is chosen.*
- *We have four lists displayed each for all cards put by the players for "Spade" ,"Diamond", "Heart" and "Clubs" respectively.*
- *We also displayed whose turn it is. If its user's turn, he's asked for an input to provide.*

*3.***Algorithm Used:**

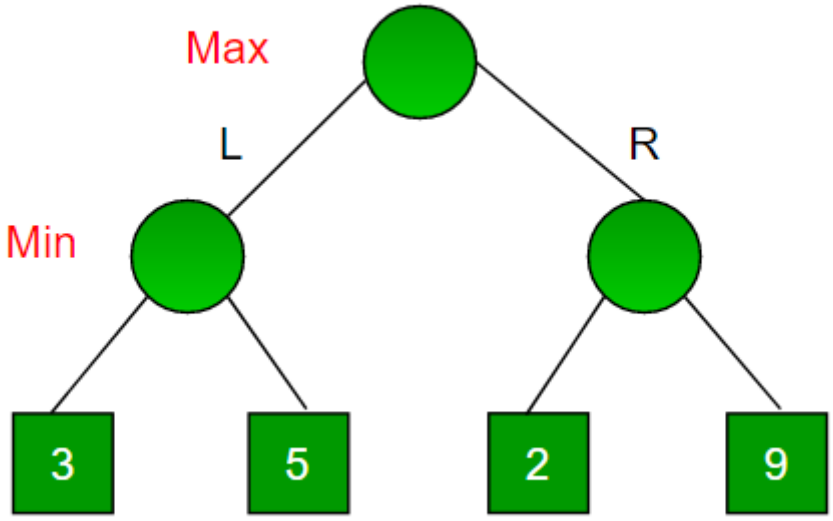
- *Algorithm used: Minimax algorithm*

- *About Minimax Algorithm:*
  - *Minimax is a kind of bacttracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally. It is widely used in two player turn-based games such as Tic-Tac-Toe, Backgammon, Mancala, Chess, etc.*

  - *In Minimax the two players are called maximizer and minimizer. The **maximizer** tries to get the highest score possible while the **minimizer** tries to do the opposite and get the lowest score possible.*

  - *Every board state has a value associated with it. In a given state if the maximizer has upper hand then, the score of the board will tend to be some positive value. If the minimizer has the upper hand in that board state then it will tend to be some negative value. The values of the board are calculated by some heuristics which are unique for every type of game.*

  - ***Example:***
    Consider a game which has 4 final states and paths to reach final state are from root to 4 leaves of a perfect binary tree as shown below. Assume you are the maximizing player and you get the first chance to move, i.e., you are at the root and your opponent at next level. ***Which move you would make as a maximizing player considering that your opponent also plays optimally?***



- *Since this is a backtracking based algorithm, it tries all possible moves, then backtracks and makes a decision.*

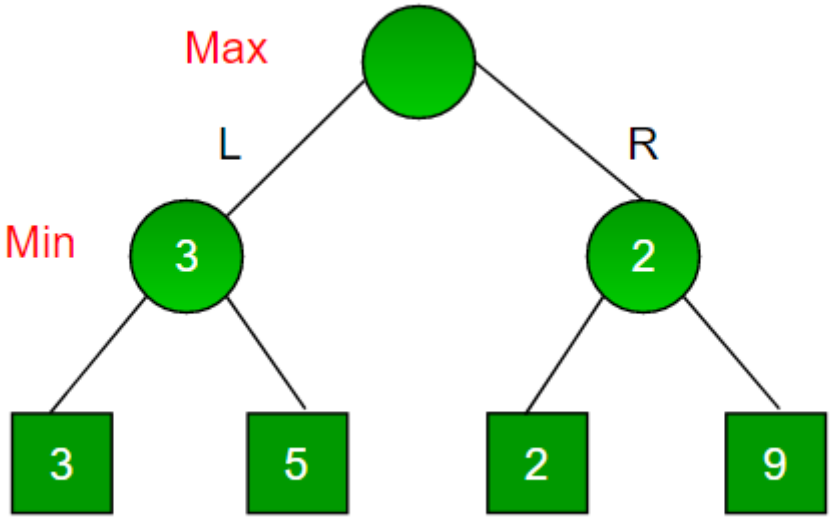
- Maximizer goes LEFT: It is now the minimizers turn. The minimizer now has a choice between 3 and 5. Being the minimizer it will definitely choose the least among both, that is 3
- Maximizer goes RIGHT: It is now the minimizers turn. The minimizer now has a choice between 2 and 9. He will choose 2 as it is the least among the two values.

Being the maximizer you would choose the larger value that is 3. Hence the optimal move for the maximizer is to go LEFT and the optimal value is 3.

Now the game tree looks like below :

- *The below tree shows two possible scores when maximizer makes left and right moves.*

Note: Even though there is a value of 9 on the right subtree, the minimizer will never pick that. We must always assume that our opponent plays optimally.



4.<u>Implementation of Minimax Algorithm:</u>

- *We have created a list containing all the valid cards for every turn so that, it would be easier for computer or user to choose from it.*

- *The above mentioned list is updated for every turn/iteration as the user/computer will have different set of cards to choose in every turn.*

- *This is where we use minimax algorithm, to choose a card from the list which is best for that turn. We've written a function that takes care of this*

.

Screenshots of minimax function:

1:

```
def minimax(turn,avail_list,useable):

        high = 13
        lower = 1
        p = 0
        t = 0
        chtype = ''
        cltype = ''
        while(high>=7):
                for i in c[turn]:
                        if(i[1]==high):
                                chtype = i[0]
                                numh = i[1]
                                p = 1
                                break
                if(p==1):
                        break
                else:
                        high -= 1


        while(lower<=7):
                for i in c[turn]:
                        if(i[1]==lower):
                                cltype = i[0]
                                numl = i[1]
                                t = 1
                                break
                if(t==1):
                        break
                else:
                        lower += 1

        low_dif = 0
        high_dif = 0

        if(chtype == 'spade'):
                l = len(spade)
                high_dif = high-spade[l-1]
        elif(chtype == 'flower'):
                l = len(flower)
                high_dif = high-flower[l-1]
        elif(chtype == 'heart'):
                l = len(heart)
                high_dif = high-heart[l-1]
```

2:

```
        high_dif = high-flower[l-1]
elif(chtype == 'heart'):
        l = len(heart)
        high_dif = high-heart[l-1]
elif(chtype == 'diamond'):
        l = len(diamond)
        high_dif = high-diamond[l-1]

if(cltype == 'spade'):
        l = len(spade)
        low_dif = spade[l-1] - lower
elif(cltype == 'flower'):
        l = len(flower)
        low_dif = flower[l-1] - lower
elif(cltype == 'heart'):
        l = len(heart)
        low_dif = heart[l-1] - lower
elif(cltype == 'diamond'):
        l = len(diamond)
        low_dif = diamond[l-1] - lower

if(high_dif > low_dif):
        if(chtype == 'spade'):
                for card in useable:
                        if(card[0]=='spade' and card[1]>=7):
                                return card
        elif(chtype == 'flower'):
                for card in useable:
                        if(card[0]=='flower' and card[1]>=7):
                                return card
        elif(chtype == 'heart'):
                for card in useable:
                        if(card[0]=='heart' and card[1]>=7):
                                return card
        elif(chtype == 'diamond'):
                for card in useable:
                        if(card[0]=='diamond' and card[1]>=7):
                                return card
else:
        if(cltype == 'spade'):
                for card in useable:
                        if(card[0]=='spade' and card[1]<7):
                                return card
        elif(cltype == 'flower'):
                for card in useable:
                        if(card[0]=='flower' and card[1]<7):
```

3:

```
elif(cltype == 'diamond'):
        l = len(diamond)
        low_dif = diamond[l-1] - lower

if(high_dif > low_dif):
        if(chtype == 'spade'):
                for card in useable:
                        if(card[0]=='spade' and card[1]>=7):
                                return card
        elif(chtype == 'flower'):
                for card in useable:
                        if(card[0]=='flower' and card[1]>=7):
                                return card
        elif(chtype == 'heart'):
                for card in useable:
                        if(card[0]=='heart' and card[1]>=7):
                                return card
        elif(chtype == 'diamond'):
                for card in useable:
                        if(card[0]=='diamond' and card[1]>=7):
                                return card
else:
        if(cltype == 'spade'):
                for card in useable:
                        if(card[0]=='spade' and card[1]<7):
                                return card
        elif(cltype == 'flower'):
                for card in useable:
                        if(card[0]=='flower' and card[1]<7):
                                return card
        elif(cltype == 'heart'):
                for card in useable:
                        if(card[0]=='heart' and card[1]<7):
                                return card
        elif(cltype == 'diamond'):
                for card in useable:
                        if(card[0]=='diamond' and card[1]<7):
                                return card

random.shuffle(useable)
if(len(useable)> 0):
        return useable[0]
```

- *Initially we find the bad cards in lower half and upper half and find out if the stretch of User's/Computer's cards are in lower or upper half and then proceed by cards in respective half.*

- *That is, if user has more cards in lower half (i.e., of smaller numbers) than in upper half (i.e. of larger numbers),He must try to decrease it and increase his chance of winning by putting lower numbered cards first.*

- *This is implemented by above function, we find out the distance and then based on it, we see range and find out if the user/computer has more lower numbered or higher numbered cards and proceed further.*