Vivekanand Education Society's Institute of Technology

Department of Computer Engineering

Subject: AI

Class :- CMPN Semester:-6 Div :- D12A

Roll No: 08	Name: Varnit Batheja		
Exp No: 06	Title: To implement a game playing algorithm		
DOP:	24-03-2022	DOS:	31-03-2022
GRADE:	LAB OUTCOMES:	SIGNATURE:	

Page No.
Date

EXPERIMENT NO 06

AIM

To implement a game playing algorithm (limited tic-tac-toe)

THEORY

Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player assuming that your opponent also plays optimaly.

It is widely used in two player two based games
Tic Tac Toe
Mancala
Chess
Backgammon

Properties of minimax algorithm

- It is complete. It will definitely find a solution (if exists) in the finite search tree.
- It is optimal if both playing optimalley
- Time complexity : $O(b^m)$: $b \rightarrow$ branching factor.
  $m \rightarrow$ maximum depth of the tree.
- Space complexity :- $Ob(m)$

Drawbacks / Limitation

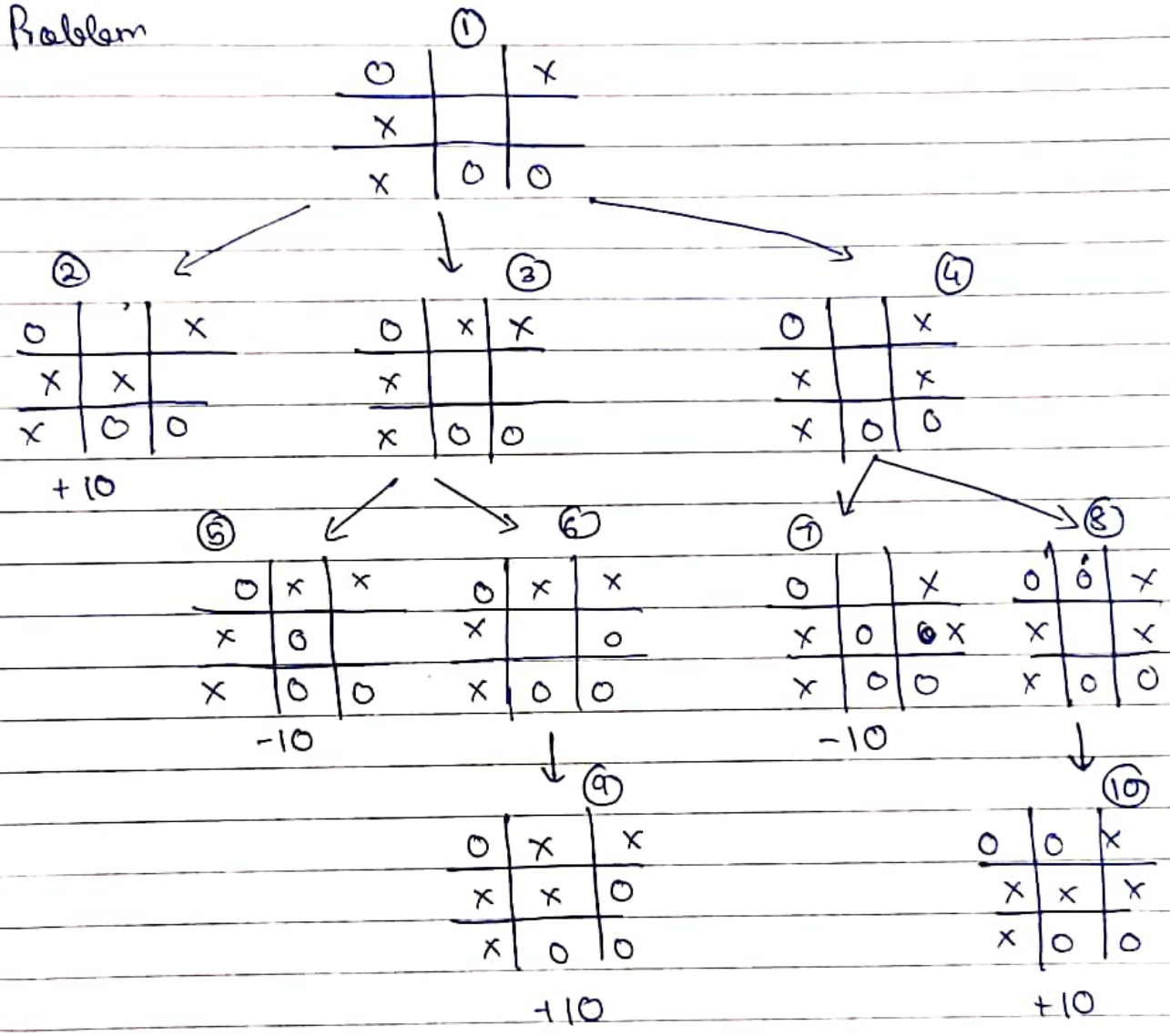
The main drawback of minimax algorithm is that it gets really slow for complex games such as chess These types of games have a huge branching factor and the player has lot of choices to decide This limitation of minimax algorithm can be.

Teacher's Sign..

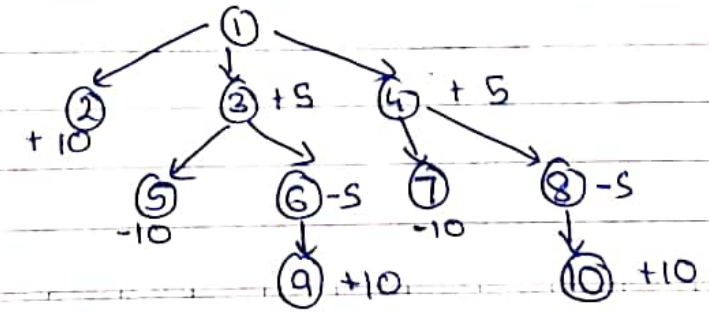
Page No.
Date

improved dey using alpha-beta pruning, which is an optimized form of minimax

Problem



Solving dey using utility values



Teacher's Sign.:

Page No.
Date

Using the utility values we find the path values and solve unlimited tic tac toe by finding the maximum value path

Path	Value
① → ②	0+10 = 10 ✓
① → ③ → ⑤	0+5-10 = -5
① → ③ → ⑥ → ⑨	0+5-5+10 = 10 ✓
① → ④ → ⑦	0+5-10 = -5
① → ④ → ⑧ → 10	0+5-5+10 = 10 ✓

Max wins 3 cases where path values are +10

CONCLUSION

Solving tic tac toe using minimax is flexible when we have the entire stack space tree which is lengthy task. we solve our limited tic tac toe problem by assigning utility values to various state and calculating path lengths / values

Teacher's Sign :

EXPERIMENT-06

Code:

```
graph = {1:[2, 3, 4], 2: [5,6,7], 3:[8,9,10], 4:[11,12,13], 5: [],
      6:[], 7: [], 8:[], 9: [], 10: [], 11: [], 12: [], 13: []}
utility = {2: 0, 3: 0, 4: 0, 5: 3, 6: 5, 7: 10, 8: 2, 9: 8, 10: 19, 11: 2, 12: 7, 13: 3}

open_list = []
closed_list = []
path = []

def min():
   op = open_list
   for i in op:
      if len(graph[i])!=0:
         for j in graph[i]:
            if utility[j]<19:
               open_list.append(j)
            else:
               closed_list.append(j)
      open_list.remove(i)
      closed_list.append(i)

def max():
   op = open_list.copy()
   for i in op:
      path = [1]
      winutility = 19
      if len(graph[i])!=0:
         for j in graph[i]:
            if utility[j]<winutility:
               path.append(i)
               path.append(j)
               print(f"Root value is : {utility[j]}")
               print(f"Path is {path}")
               exit()
               closed_list.append(j)
               break
            else:
               open_list.append(j)
```

```
        open_list.remove(i)
        closed_list.append(i)

def minmax():
    open_list.append(1)
    c=0
    while len(open_list)!=0:
        if c==0:
            min()
            c=1
        else:
            max()
            c=0

minmax()
```

Output:

```
Root value is : 3
Path is [1, 2, 5]


...Program finished with exit code 0
Press ENTER to exit console.
```