Game Playing Agent — Minimax — Alpha-Beta Pruning

Systematic adversarial search can lead to savings in terms of pruning of sub-trees resulting in lesser node evaluations

Ankur Singh
MTECH CSE (2020-22)
IIIT Vadodara
Gandhinagar, India
202061001@iiitvadodara.ac.in

Anand Mundhe
MTECH CSE (2020-22)
IIIT Vadodara
Gandhinagar, India
202061006@iiitvadodara.ac.in

I. Problems and Solutions

1) What is the size of the game tree for Noughts and Crosses? Sketch the game tree.

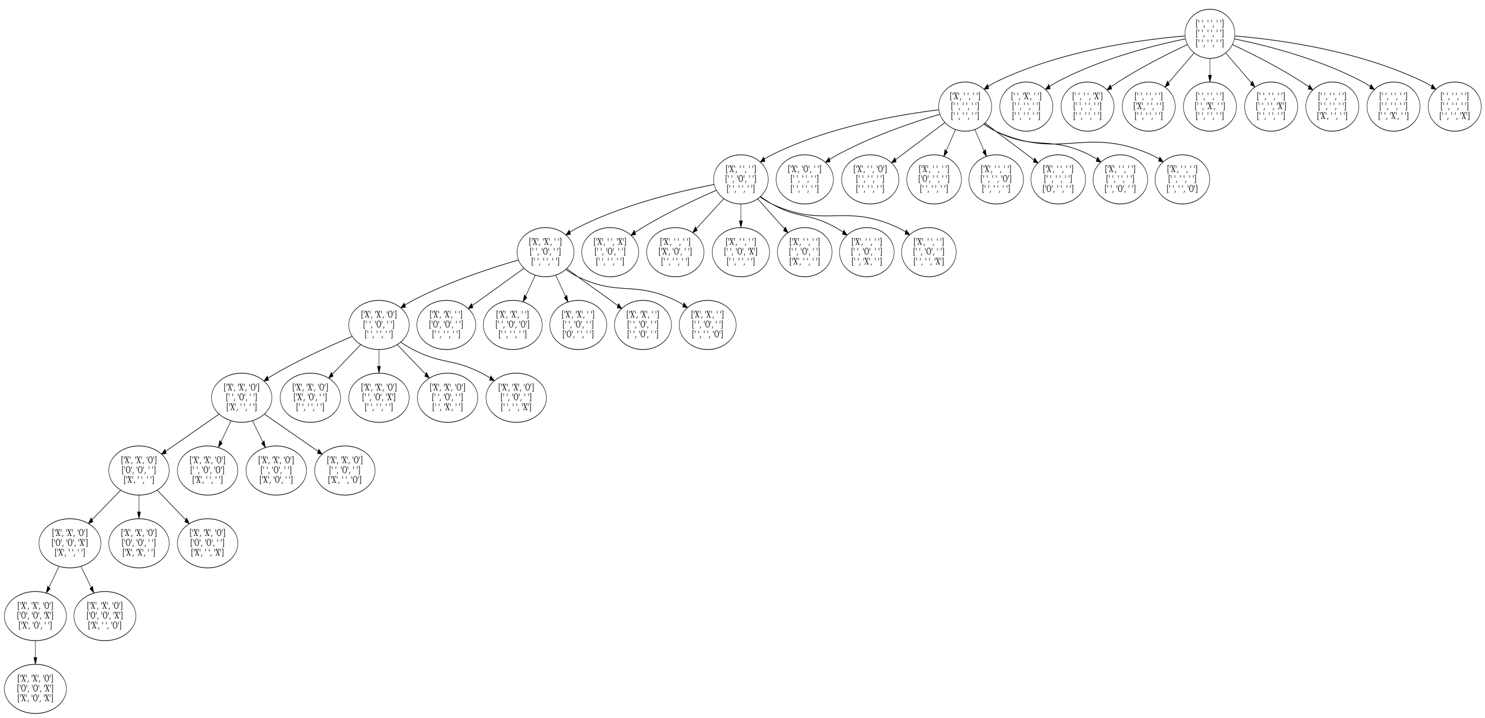

Fig. 1. Game tree of Noughts and Crosses

2) Read about the game of Nim (a player left with no move losing the game). For the initial configuration of the game with three piles of objects as shown in Figure, show that regardless of the strategy of player-1, player-2 will always win. Try to explain the reason with the MINIMAX value backup argument on the game tree.

Fig. 2. Pile of coins in game of Nim

The Game of Nim is defined by rules given as- Given a number of piles and each pile contains some number of coins. In each turn, a player can choose any number of coins(at least one) from the pile. The player with no move loses the game and the one who takes the last coin is the winner. The game depends on the factors-

a) The player who starts first.
b) The initial configurtion of the piles

To understand the winning and losing condition of this game, it is necessary to get familiar about **Nim-Sum**. A Nim-Sum is the cumulative XOR value of the number of coins in each pile at any point in the game. The following properties of XOR sum makes the game interesting

a) If the XOR sum is zero, then it is impossible to make the XOR sum stay zero by single reduction of a number.
b) If the XOR sum is non zero, there exist at least one way by which the XOR sum can be made zero by reducing the number.

It is clear that the optimal strategy for each player is to make the Nim-Sum for the opponent zero in each turn. The given problem contains 10 coins in pile 1, 7 coins in pile 2 and 9 coins in pile 3. 7 XOR 10 XOR 9 gives 4 which is a non zero value. Thus if player 2 is taking the first move then regardless the strategy adopted by player 1, player 2 will win because whatever move the player 1 takes it will definitely make the nim sum non zero only. [2]

3) Implement MINIMAX and alpha-beta pruning agents. Report on number of evaluated nodes for Noughts and Crosses game tree.

```
    #Pruned AI
import math
class Node:
    def __init__(self,state=[' ']*9,p1='X',p2='O',alpha=-math.inf,beta=-math.inf):
        self.state=state
        self.p=[p1,p2]
        self.child=None
        self.children=[]
        self.ready=False
        self.alpha=alpha
        self.beta=beta
    def winner(self,Type):
        #Player 1 maximizer
        x=lambda t:-1 if Type==0 else 1
        #Row win for given player number
        for i in range(3):
            for j in range(3):
                if self.state[i*3+j]!=self.p[Type]:break
                elif j==2:return x(Type)
        #Col win
        for i in range(3):
            for j in range(3):
                if self.state[j*3+i]!=self.p[Type]:break
                elif j==2:return x(Type)
        #Diagonal win
        if self.state[4]==self.p[Type]:
            for i in [0,8]:
                if self.state[i]!=self.p[Type]:break
                elif i==8:return x(Type)
            for i in [2,6]:
                if self.state[i]!=self.p[Type]:break
                elif i==6:return x(Type)
        #no winner and no move mean Tie
        if self.remaining_moves()==0:return 0
        #otherwise None, not final state
        return None
    def remaining_moves(self):
        return self.state.count(' ')
    def getChildren(self):
        out=[]
        for i in range(len(self.state)):
            if self.state[i]==' ':
                out.append(i)
        return out
class Agent:
    def __init__(self):
        self.NodesCount=0
    def mm(self,node,Type):
        #Type 0 min player
        #Type 1 max player
        if Type==0:cond=lambda a,b: a>b if a!=None else True
        else:cond=lambda a,b: a<b if a!=None else True
        #Check if alredy win
        tmp=node.winner((Type+1)%2)
        if tmp!=None:
            return node,tmp*(node.remaining_moves()+1)   #Utility
        #Not win yet so check children
        tnode=None
        tmp=None
        for i in node.getChildren():
            #get child and make move
            node.state[i]=node.p[Type]
            #run same for child
            a,b=self.mm(Node(node.state[:],node.p[0],node.p[1],node.alpha,node.beta),(Type+1)%2)
            #Node count increases
            self.NodesCount+=1
            #Append child in node's children
            node.children.append(a)
            #b is Utility and is none when at
```

```
leaf node (Win Draw or Lose)
            if b!=None and cond(tmp,b):
                #Assign alpha and beta values to node accordingly
                if Type==0:
                    if node.beta>b:node.beta=b
                else:
                    if node.alpha<b:node.alpha=b
                tmp=b
                tnode=a
            #Undo the last move
            node.state[i]=' '
            #Prune next children if alpha>beta
            if node.alpha>node.beta:break
        node.child=tnode
        node.ready=True
        return node,tmp

        #Driver for Pruned AI
class Game:
    #initilise game
    def __init__(self,p1,p2):
        self.node=Node([' ']*9,p1=p1,p2=p2)
    #Print state of game
    def pstate(self):
        for i in range(3):
            for j in range(3):
                print(self.node.state[i*3+j],end="\t|\t")
            print()
    #play a move either player 0 or 1
    def play(self,Type):
        #Type 0 min player, User player
        #Type 1 max player, Computer player
        #User input variable
        ip=-1
        if Type==0:
            #Check if ip is valid, and take input till valid input is there
            while not ip in self.node.getChildren():
                ip=int(input("Enter next move : "+str(self.node.getChildren())))
                if(not ip in self.node.getChildren()):print("Invalid move")
            #Apply input to state
            self.node.state[ip]=self.node.p[Type]
        #Computer's move
        else:
            #Initially agent is not executed, so run once
            if not self.node.ready:
                #return node with children and utility value
                obj=Agent()
                self.node.alpha=-math.inf
                self.node.beta=+math.inf
                self.node,win=obj.mm(self.node,1)
                print("Smart AI is ready with total node exploration :",obj.NodesCount," With result : ",win)
                #Make 1 move by heading to child
                self.node=self.node.child
                #Now agent is ready
                self.node.ready=True
                return
            #from second time onward, Check children for user move's node
            for i in self.node.children:
                #check till user move node is found
                if i.state==self.node.state:
                    #from that user move node, if move is possible then make move
                    if i.child!=None:
                        self.node=i.child
                        #again agent is still ready
                        self.node.ready=True
                    return
    #Execution of game from here
    def drive(self):
        #initial player -1
        turn=-1
        #Play untill no more moves are possible or node is None
        while self.node!=None and self.node.remaining_moves()!=0:
            #This shows utility value for final state
            print("Player 1 win : ",self.node.winner(0))
            print("Player 2 win : ",self.node.winner(1))
            #display current state
            self.pstate()
            #User's move
            self.play((turn+1)%2)
            #Check if user win
            if self.node.winner(0)==-1:
                print("You win")
                break
            #Check if game tie
            if self.node.winner(0)==0:
                print("Game is tie")
                break

            #Computer's move
            self.play((turn+2)%2)
            #Check if computer wins
            if self.node!=None and self.node.winner(1)==1:
                print("Computer wins...")
                break
            #Check if no child mean computer cant make move
            if self.node==None:break
            #Check ig game tie
            if self.node.winner(1)==0:
                print("Game is tie")
                break
        #Show's final states when game is finished
        if self.node!=None:
            self.pstate()
            print("Player 1 win : ",self.node.winner(0))
            print("Player 2 win : ",self.node.winner(1))
        #When no child availabe, computer is stuck
        else:
            print("Computer cannot make move")

        Game(0,1).drive()
```

```
    #Pruned AI computer vs computer
class Game:
    #initilise game
    def __init__(self,p1,p2):
        self.node=Node([' ']*9,p1=p1,p2=p2)
    #Print state of game
```

```
Player 1 win :  None
Player 2 win :  None
        |               |               |
        |               |               |
        |               |               |
Enter next move : [0, 1, 2, 3, 4, 5, 6, 7, 8]0
Smart AI is ready with total node exploration : 10556 With result :  0
Player 1 win :  None
Player 2 win :  None
0       |               |               |
        |       1       |               |
        |               |               |
Enter next move : [1, 2, 3, 5, 6, 7, 8]8
Player 1 win :  None
Player 2 win :  None
0       |       1       |               |
        |       1       |               |
        |               |       0       |
Enter next move : [2, 3, 5, 6, 7]7
Player 1 win :  None
Player 2 win :  None
0       |       1       |               |
        |       1       |               |
1       |       0       |       0       |
Enter next move : [2, 3, 5]2
Player 1 win :  None
Player 2 win :  None
0       |       1       |       0       |
        |       1       |       1       |
1       |       0       |       0       |
Enter next move : [3]3
Game is tie
0       |       1       |       0       |
0       |       1       |       1       |
1       |       0       |       0       |
Player 1 win :  0
Player 2 win :  0
```

Fig. 3. Output of game

```python
    def pstate(self):
        for i in range(3):
            for j in range(3):
                print(self.node.state[i*3+j], end="\t|\t")
            print()
    #play a move either player 0 or 1
    def play(self,Type):
        #Type 0 min player, User player
        #Type 1 max player, Computer player
        #User input variable
        ip=-1
        if Type==0:
            #Check if ip is valid, and take input till valid input is there
            while not ip in self.node.getChildren():
                ip=int(input("Enter next move : "+str(self.node.getChildren())))
                if(not ip in self.node.getChildren()):print("Invalid move")
            #Apply input to state
            self.node.state[ip]=self.node.p[Type]
        #Computer's move
        else:
            #Initially agent is not executed, so run once
            if not self.node.ready:
                #return node with children and utility value
                obj=Agent()
                self.node.alpha=-math.inf
                self.node.beta=math.inf
                self.node,win=obj.mm(self.node,0)
                print("Smart AI is ready with total node exploration :",obj.NodesCount)
                #Make 1 move by heading to child
                self.node=self.node.child
                #Now agent is ready
                self.node.ready=True
                return
            #from second time onward, Check children for user move's node
            self.node=self.node.child
#Execution of game from here
def drive(self):
    #initial player -1
    turn=-1
    #Play untill no more moves are possible or node is None
    while self.node!=None and self.node.remaining_moves()!=0:
        #This shows utility value for final state
        print("Player 1 win : ",self.node.winner(0))
        print("Player 2 win : ",self.node.winner(1))
        #display current state
        self.pstate()
        #Computer 1's move
        self.play((turn+2)%2)
        #Check if computer win
        if self.node.winner(0)==-1:
            print("You win")
            break
        #Check if game tie
        if self.node.winner(0)==0:
            print("Game is tie")
            break


        #This shows utility value for final state
        print("Player 1 win : ",self.node.winner(0))
        print("Player 2 win : ",self.node.winner(1))
        #display current state
        self.pstate()
        #Computer 2's move
        self.play((turn+2)%2)
        #Check if computer wins
        if self.node!=None and self.node.winner(1)==1:
            print("Computer wins...")
            break
        #Check if no child mean computer cant make move
        if self.node==None:break
        #Check ig game tie
        if self.node.winner(1)==0:
            print("Game is tie")
            break
    #Show's final states when game is finished
    if self.node!=None:
        self.pstate()
        print("Player 1 win : ",self.node.winner(0))
        print("Player 2 win : ",self.node.winner(1))
    #When no child availabe, computer is stuck
```

```python
        else:
            print("Computer cannot make move")

        Game(0,1).drive()

```

```python
    #Pruned AI vs unpruned AI
import math
class Node:
    def __init__(self,state=[' ']*9,p1='X',p2='O',alpha=-math.inf,beta=-math.inf):
        self.state=state
        self.p=[p1,p2]
        self.child=None
        self.children=[]
        self.ready=False
        self.alpha=alpha
        self.beta=beta
    def winner(self,Type):
        #Player 1 maximizer
        x=lambda t:-1 if Type==0 else 1
        #Row win for given player number
        for i in range(3):
            for j in range(3):
                if self.state[i*3+j]!=self.p[Type]:break
                elif j==2:return x(Type)
        #Col win
        for i in range(3):
            for j in range(3):
                if self.state[j*3+i]!=self.p[Type]:break
                elif j==2:return x(Type)
        #Diagonal win
        if self.state[4]==self.p[Type]:
            for i in [0,8]:
                if self.state[i]!=self.p[Type]:break
                elif i==8:return x(Type)
            for i in [2,6]:
                if self.state[i]!=self.p[Type]:break
                elif i==6:return x(Type)
        #no winner and no move mean Tie
        if self.remaining_moves()==0:return 0
        #otherwise None, not final state
        return None
    def remaining_moves(self):
        return self.state.count(' ')
    def getChildren(self):
        out=[]
        for i in range(len(self.state)):
            if self.state[i]==' ':
                out.append(i)
        return out
class AgentUnprune:
    def __init__(self):
        self.NodesCount=0
    def mm(self,node,Type):
        #Type 0 min player
        #Type 1 max player
        if Type==0:cond=lambda a,b: a>b if a!=None else True
        else:cond=lambda a,b: a<b if a!=None else True
        #Check if alredy win
        tmp=node.winner((Type+1)%2)
        if tmp!=None:
            return node,tmp*(node.remaining_moves()+1)   #Utility
        #Not win yet so check children
        tnode=None
        tmp=None
        for i in node.getChildren():
```

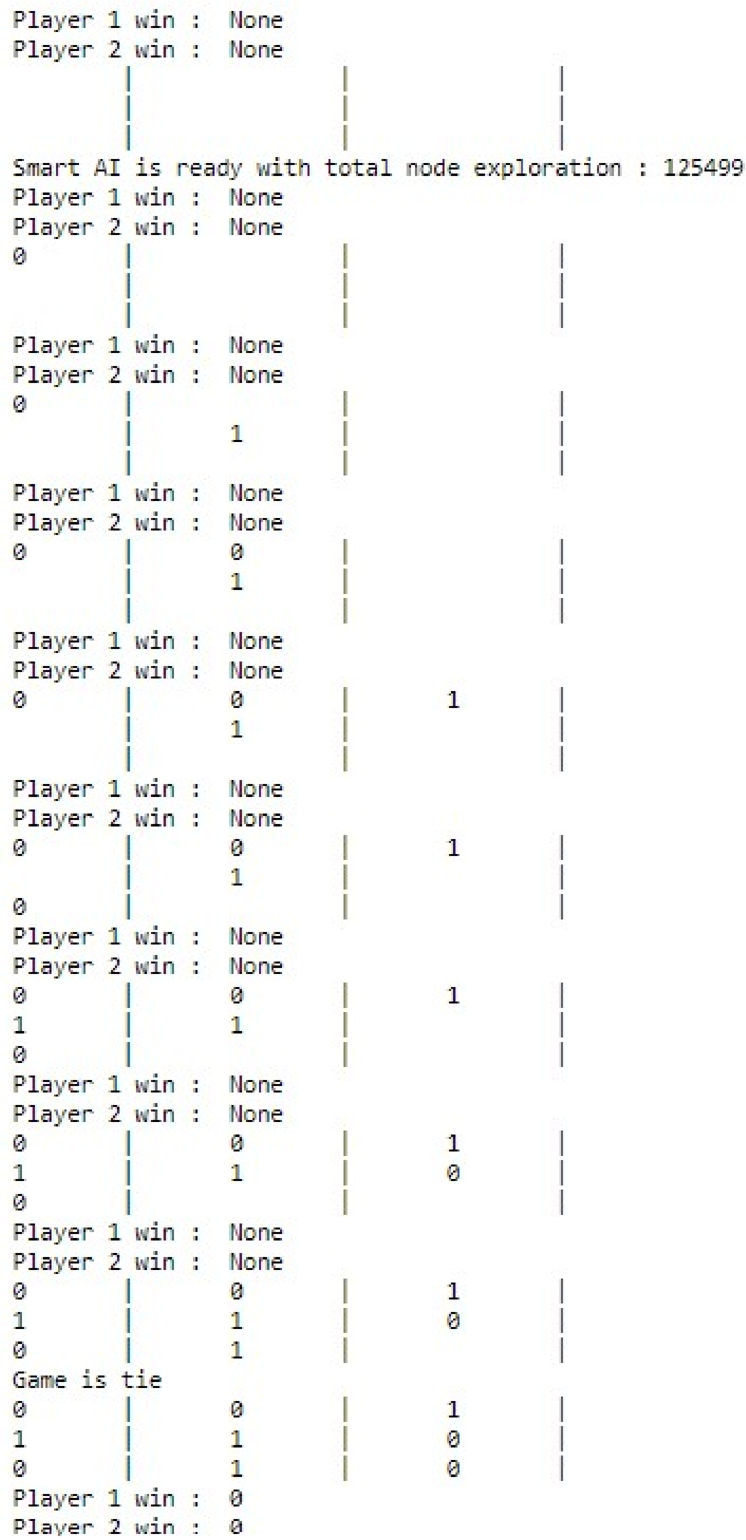

Fig. 4. Output of game

```python
            #get child and make move
            node.state[i]=node.p[Type]
            #run same for child
            a,b=self.mm(Node(node.state[:],node.p[0],node.p[1],node.alpha,node.beta),(Type+1)%2)
            #Node count increases
            self.NodesCount+=1
            #Append child in node's children
            node.children.append(a)
            #b is Utility and is none when at leaf node (Win Draw or Lose)
            if b!=None and cond(tmp,b):
                tmp=b
                tnode=a
            node.state[i]=' '
        node.child=tnode
        node.ready=True
        return node,tmp

    #Driver game for Ai vs pruned AI
class Game:
    #initilise game
    def __init__(self,p1,p2):
        self.node=Node([' ']*9,p1=p1,p2=p2)
    #Print state of game
    def pstate(self):
        for i in range(3):
            for j in range(3):
                print(self.node.state[i*3+j], end="\t|\t")
            print()
    #play a move either player 0 or 1
    def play(self,Type):
        #Type 0 min player, User player
        #Type 1 max player, Computer player
        #Computer's input variable
        if Type==0:
            #Initially agent is not executed, so run once
            if not self.node.ready:
                #return node with children and utility value
                obj=AgentUnprune()
                self.node1,win=obj.mm(self.node,0)
                print("Smart AI is ready with total node exploration :",obj.NodesCount)
                #Make 1 move by heading to child
                self.node=self.node1.child
                self.node1=self.node1.child
                #Now agent is ready
                self.node.ready=False
                return
            #from second time onward, Check children for user move's node
            for i in self.node1.children:
                #check till user move node is found
                if i.state==self.node.state:
                    #from that user move node, if move is possible then make move
                    if i.child!=None:
                        self.node=i.child
                        self.node1=i.child
                        #again agent is still ready
                        self.node.ready=True
                    return
        #Computer's move
        else:
            #Initially agent is not executed, so run once
            if not self.node.ready:
                #return node with children and utility value
                obj=Agent()
                self.node.alpha=-math.inf
                self.node.beta=math.inf
                self.node2,win=obj.mm(self.node,1)
                print("Smart AI is ready with total node exploration :",obj.NodesCount)
                #Make 1 move by heading to child
                self.node=self.node2.child
                self.node2=self.node2.child
                #Now agent is ready
                self.node.ready=True
                return
            #from second time onward, Check children for user move's node
            for i in self.node2.children:
                #check till user move node is found
                if i.state==self.node.state:
                    #from that user move node, if move is possible then make move
                    if i.child!=None:
                        self.node=i.child
                        self.node2=i.child
                        #again agent is still ready
                        self.node.ready=True
                    return
    #Execution of game from here
    def drive(self):
        #initial player -1
        turn=-1
        #Play untill no more moves are possible or node is None
        while self.node!=None and self.node.remaining_moves()!=0:
            #This shows utility value for final state
            print("Player 1 win : ",self.node.winner(0))
            print("Player 2 win : ",self.node.winner(1))
            #display current state
            self.pstate()
            #User's move
            self.play((turn+1)%2)
            #Check if user win
            if self.node.winner(0)==-1:
                print("Computer 1 win")
                break
            #Check if game tie
            if self.node.winner(0)==0:
                print("Game is tie")
                break

            #This shows utility value for final state
            print("Player 1 win : ",self.node.winner(0))
            print("Player 2 win : ",self.node.winner(1))
            #display current state
            self.pstate()
            #Computer's move
            self.play((turn+2)%2)
            #Check if computer wins
            if self.node!=None and self.node.winner(1)==1:
                print("Computer 2 wins...")
                break
```

```
            #Check if no child mean computer cant make move
            if self.node==None:break
            #Check ig game tie
            if self.node.winner(1)==0:
                print("Game is tie")
                break
        #Show's final states when game is finished
        if self.node!=None:
            self.pstate()
            print("Player 1 win : ",self.node.winner(0))
            print("Player 2 win : ",self.node.winner(1))
        #When no child availabe, computer is stuck
        else:
            print("Computer cannot make move")

    Game(0,1).drive()
```

```
Player 1 win :  None
Player 2 win :  None
         |                 |                 |
         |                 |                 |
         |                 |                 |
Smart AI is ready with total node exploration : 549945
Player 1 win :  None
Player 2 win :  None
0        |                 |                 |
         |                 |                 |
         |                 |                 |
Smart AI is ready with total node exploration : 10556
Player 1 win :  None
Player 2 win :  None
0        |                 |                 |
         |        1        |                 |
         |                 |                 |
Player 1 win :  None
Player 2 win :  None
0        |        0        |                 |
         |        1        |                 |
         |                 |                 |
Player 1 win :  None
Player 2 win :  None
0        |        0        |        1        |
         |        1        |                 |
         |                 |                 |
Player 1 win :  None
Player 2 win :  None
0        |        0        |        1        |
         |        1        |                 |
0        |                 |                 |
Player 1 win :  None
Player 2 win :  None
0        |        0        |        1        |
1        |        1        |                 |
0        |                 |                 |
Player 1 win :  None
Player 2 win :  None
0        |        0        |        1        |
1        |        1        |        0        |
0        |                 |                 |
Player 1 win :  None
Player 2 win :  None
0        |        0        |        1        |
1        |        1        |        0        |
0        |        1        |                 |
Game is tie
0        |        0        |        1        |
1        |        1        |        0        |
0        |        1        |        0        |
Player 1 win :  0
Player 2 win :  0
```

Fig. 5. Output of game

4) Using recurrence relation show that under perfect ordering of leaf nodes, the alpha-beta pruning time complexity is $O(b^{m/2})$, where b is the effective branching factor and m is the depth of the tree.

   Let $T(m)$ be the search complexity for depth m. Normally under minimax algorithm without alpha beta pruning, the recurrence relation will be

   $$T(m) = b.T(m-1) + c$$

   The time complexity would be $O(b^m)$. For alpha-beta pruning the recurrence relation would be

   $$T(m) = T(m-1) + (b-1).T(m-2) + c$$

   because it is necessary to know the value of first child i.e. $T(m-1)$ and this child has sub-tree of height $m-1$. For the other children, it is required to compute value at one of their children at depth $m-2$. Solving this,

   $$T(m) = T(m-2)+(b-1).T(m-3)+(b-1).T(m-2)+c$$

   $$T(m) = b.T(m-2) + (b-1).T(m-3) + c$$

   It is evident that $T(m-3) < T(m-2)$,so:

   $$T(m) < (2b-1).T(m-2)$$

   $$T(m) < 2b.T(m-2)$$

   i.e., the branching factor every two levels is less than 2b which means that the effective branching factor is less than $\sqrt{2b}$. This is not too far off the asymptotic upper bound of $\sqrt{b}+1)^{m+1}$ hence, giving the final time complexity as $O(b^{m/2})$ This is a substantial improvement. It does not affect the final result. With perfect ordering, it doubles the depth of search. [3]

REFERENCES

[1] Artificial Intelligence: a Modern Approach, Russell and Norvig (Fourth edition)

[2] Combinatorial Game Theory — Set 2 (Game of Nim). https://www.geeksforgeeks.org/combinatorial-game-theory-set-2-game-nim/

[3] Best-Case Analysis of Alpha-Beta Pruning http://www.cs.utsa.edu/ bylander/cs5233/a-b-analysis.pdf