**Batch: B2 Roll No.: 1411099**

**Experiment / assignment / tutorial No. 6**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title:** Implementation of Min-Max algorithms |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Objective:** Implementation of minimax algorithm

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **Course Outcome** | **After successful completion of the course students should be able to** |
| **CO2** | Analyse and solve problems for goal based agent architecture (searching and planning algorithms). |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **“Artificial Intelligence: a Modern Approach” by Russel and Norving, Pearson education Publications**
2. **“Artificial Intelligence” By Rich and knight, Tata Mcgraw Hill Publications**
3. [**www.cs.sfu.ca/CourseCentral/310/oschulte/mychapter5.pdf**](http://www.cs.sfu.ca/CourseCentral/310/oschulte/mychapter5.pdf)
4. [**http://cs.lmu.edu/~ray/notes/asearch/**](http://cs.lmu.edu/~ray/notes/asearch/)
5. **www.cs.cornell.edu/courses/cs4700/2011fa/.../06\_adversarialsearch.pdf**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Pre Lab/ Prior Concepts:** Two/Multi player Games and rules, state-space tree, searching algorithms and their analysis properties

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Historical Profile: -** The game playing has been integral part of human life. The multiplayer games are competitive environment in which everyone tries to gain more points for himself and wishes the opponent to gain minimum.

The game can be represented in form of a state space tree and one can follow the path from root to some goal node, for either of the player.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

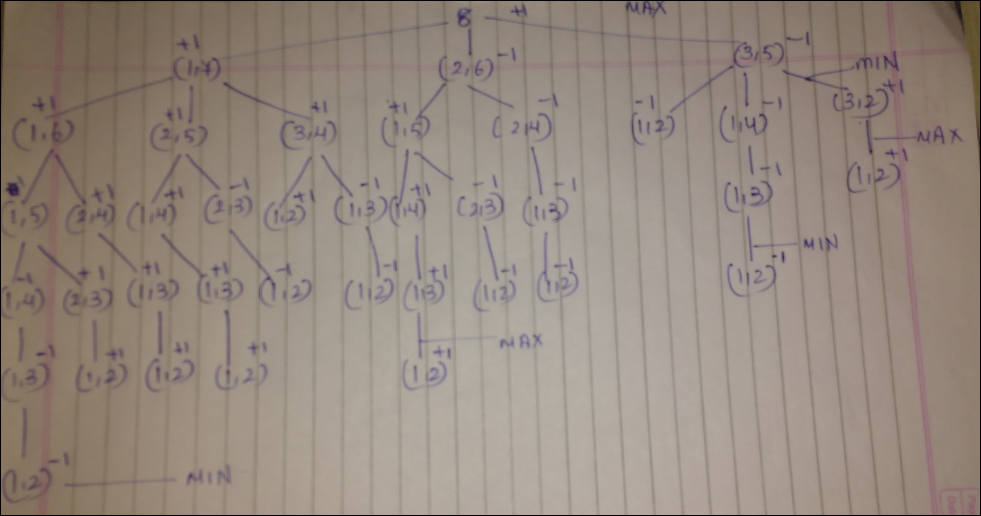
**New Concepts to be learned:** Adversarial search, minmax algorithm, minmax pruning,

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Adversarial Search:-**

A minimax algorithm] is a recursive algorithm for choosing the next move in an n-player game, usually a two-player game. A value is associated with each position or state of the game. This value is computed by means of a position evaluation function and it indicates how good it would be for a player to reach that position. The player then makes the move that maximizes the minimum value of the position resulting from the opponent's possible following moves. If it is A's turn to move, A gives a value to each of his legal moves.

A possible allocation method consists in assigning a certain win for A as +1 and for B as −1.An alternative is using a rule that if the result of a move is an immediate win for A it is assigned positive infinity and, if it is an immediate win for B, negative infinity. The value to A of any other move is the minimum of the values resulting from each of B's possible replies. For this reason, A is called the maximizing player and B is called the minimizing player, hence the name minmax algorithm.

****

**Minimax algorithm:**

The minimax algorithm does exactly that: it finds the move that minimizes the maximum utility the opponent can obtain. You can call it "maximin" if you'd like: maximize the minimum utility we (not the opponent) can obtain. They are the same algorithm, just the order of the steps (maximize or minimize) is different. First, we consider all of our possible moves. If a move is a final move (winning/losing/tie), then we calculate its utility directly (1.0 for a win, -1.0 for a loss, 0.0 for a tie). If it's not a final move, we have to look further to calculate its utility. We choose one move from which to "look further," and consider all the possible moves the opponent might make. If the opponent wins, the utility of the opponent's move is -1.0. Of all the possible moves the opponent might make, we find the worst (minimum) utility. That utility is the value of our move. Of all our possible moves, we choose the one that has the maximum utility; it is the maximum of the minimum utilities generated by the opponent's moves. We are assuming the opponent is also choosing the best move, so when we choose our move, we want to find the move that performs best (max) assuming the opponent also performs best (min; i.e., max for the opponent, min for us).

**Chosen Problem:**

**Split a given number into unequal parts. The one who ends up with a number that cannot be split further wins.**

**Solution of chosen Problem:**

**CODE:**

import java.io.\*;

import java.util.\*;

class Node{

ArrayList<Node> left;

ArrayList<Node> right;

int left1;

int right1;

int value;  //to store whether its max or min

int val;    //to store its value

int level;  //to store the depth

Node parent;

boolean visited;

Node(int l,int r,int val,Node par,int le){

left1 = l;

right1 = r;

value = val;

parent = par;

level = le;

}

}

class FirstNode{

ArrayList<Node> nodes;

int data;

boolean visited;

FirstNode(int x)

    {

data = x;

    }

}

class MinMaxGame

    {

public static void main(String[] args) throws IOException

    {

    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

    int num = Integer.parseInt(br.readLine());

    FirstNode firstNode = new FirstNode(num);

ArrayList<Node> old = new ArrayList<>();

ArrayList<Node> firstLevel = new ArrayList<>();

int value = 1;

int depth = 0,k=0;

for(int i =1;i<=num/2;i++)

    {

Node temp = new Node(i,num-i,value,null,1);

old.add(temp);

firstLevel.add(temp);

    }

ArrayList<Node> lastLevel = new ArrayList<>();

int level = 1;

while(!old.isEmpty())

    {

k++;

level++;

ArrayList<Node> newLevel = new ArrayList<>();

if(value == 1)

    {

value = -1;

}

else

{

value = 1;

}

for(Node n : old)

    {

ArrayList<Node> leftChild = new ArrayList<>();

ArrayList<Node> rightChild = new ArrayList<>();

for(int i = 1;i<=n.left1/2;i++)

    {

if(i!=n.left1-i)

    {

Node temp = new Node(i,n.left1-i,value,n,level);

leftChild.add(temp);

newLevel.add(temp);

    }

    }

for(int i = 1;i<=n.right1/2;i++)

    {

if(i!=n.right1-i)

    {

Node temp = new Node(i,n.right1-i,value,n,level);

rightChild.add(temp);

newLevel.add(temp);

    }

    }

n.left = leftChild;

n.right = rightChild;

    }

if(newLevel.isEmpty()){

depth = k;

}

old = newLevel;

}

ArrayList<Node> solution = new ArrayList<>();

while(depth!=0)

    {

old = firstLevel;

level = 0;

while(level<=depth)

    {

ArrayList<Node> newLevel = new ArrayList<>();

for(Node n : old)

    {

ArrayList<Node> right = n.right;

ArrayList<Node> left = n.left;

for(Node n1 : right)

    {

newLevel.add(n1);

    }

for(Node n2 : left)

    {

newLevel.add(n2);

    }

    }

old = newLevel;

level++;

if(level==depth)

    {

for(Node n : newLevel)

    {

if(n.right.isEmpty() && n.left.isEmpty())

    {

n.val = n.value;

if(n.val==1)

    {

    solution.add(n);

    }

    }

else

    {

if(n.value==1)

    {

ArrayList<Node> temp1 = n.left;

ArrayList<Node> temp2 = n.right;

int max = -1;

for(Node n1: temp1)

    {

if(n1.val>=max)

    {

max = n1.val;

    }

    }

for(Node n1: temp2)

    {

if(n1.val>=max)

    {

max = n1.val;

    }

    }

n.val = max;

    }

else

    {

ArrayList<Node> temp1 = n.left;

ArrayList<Node> temp2 = n.right;

int min = 1;

for(Node n1: temp1)

    {

if(min>n1.val)

    {

min = n1.val;

    }

    }

for(Node n1: temp2)

    {

if(min>n1.val)

    {

min = n1.val;

    }

    }

n.val = min;

    }

    }

    }

    }

    }

depth--;

    }

for(Node n: firstLevel)

    {

ArrayList<Node> temp1 = n.left;

ArrayList<Node> temp2 = n.right;

int max = -1;

for(Node n1: temp1)

    {

if(n1.val>=max)

    {

max = n1.val;

    }

    }

for(Node n1: temp2)

    {

if(n1.val>=max)

    {

max = n1.val;

    }

    }

n.val = max;

    }

old = new ArrayList<>();

old = firstLevel;

int winOrLose = 0;

int choice;

while(!old.isEmpty())

    {

System.out.println("Choose one of the following");

int i = 1;

    for(Node n : old)

        {

    System.out.print(i+")  " +"["+n.left1+","+n.right1+"]   ");

    i++;

        }

    System.out.println();

    choice = Integer.parseInt(br.readLine());

    Node temp = old.get(choice-1);

    ArrayList<Node> newLevel = new ArrayList<>();

    ArrayList<Node> rightChild = temp.right;

    ArrayList<Node> leftChild = temp.left;

    for(Node n1 : rightChild)

        {

    newLevel.add(n1);

        }

    for(Node n1 : leftChild)

        {

    newLevel.add(n1);

        }

    if(newLevel.isEmpty())

        {

    winOrLose = old.get(0).value;

        }

    old = newLevel;

}

if(winOrLose==-1)

  {

    System.out.println("Agent Lost");

  }

else

  {

    System.out.println("Agent Won");

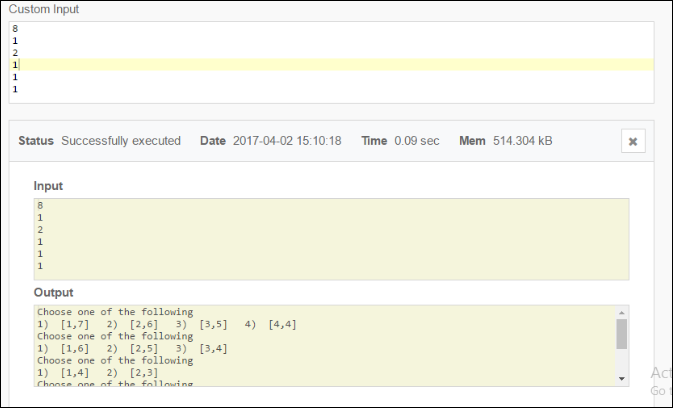
  }

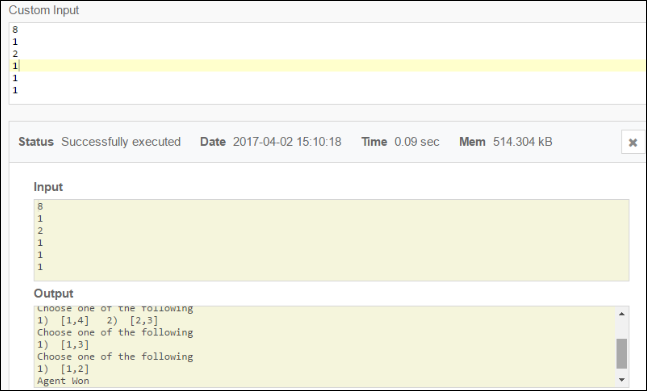
    }

}

**OUTPUT:**

**Consider Number 8:**

****

****

**Post Lab objective Questions:**

1. **Which search is equal to minmax search but eliminates the branches that can’t influence the final decision?**
   1. Breadth-first search
   2. Depth first search
   3. Alpha-beta pruning
   4. None of the above

**Answer: C**

1. **Which values are independent in minmax search alogirthm?**
   1. Pruned leaves x and y
   2. Every states are dependant
   3. Root is independent
   4. None of the above

**Answer: A**

**Post Lab Subjective Questions:**

* + - 1. **Explain the concept of adversarial search**
* In computer science, a search algorithm is an algorithm for finding an item with specified properties among a collection of items. The items may be stored individually as records in a database; or may be elements of a search space defined by a mathematical formula or procedure.

Adversarial search in Game playing : “ In which we examine the problems that arise when we try to plan ahead in a world where other agents are planning against us”. The components of a search problem included the following:

**Initial state**

some description of the agent's starting situation

**Possible actions**

the set of actions (such as chess moves) available to the agent, also called "applicable" actions; the possible actions depend on the state

**Transition model**

some way of figuring out what an action does; in other words, a resultOf(state, action) function which returns a state; the transition model defines a state space, which takes the form of a directed graph (vertices are states, edges are actions)

Note that we have left out "Goal criteria" and "Path cost." In order to support adversarial search, we add the following:

**Players**

a list of players (and a switch\_player(player) function that switches to the player following player

**Terminal tests**

a function or functions that test for final states (winning/losing/tied states)

**Utility function**

a function utility(state, player) that returns a real number (or integer) representing the value to player of state; presumably, winning states have the highest utility, losing states the lowest utility, tied states have zero utility, and states that are not terminal have a utility calculated by looking at the utilities of following states.

The way to do this is to treat the opponent's positive utility as our negative utility. So if the opponent makes a winning move, that move is worth 1.0 to the opponent but worth -1.0 to us (a tie game has utility 0.0 for both players). We want to find the move that maximizes our utility (best case, 1.0) and minimizes the opponent's utility (best case, -1.0).

* + - 1. **Explain how alpha-beta pruning improves memory efficiency of algorithm**
* The algorithm maintains two values, alpha and beta, which represent the maximum score that the maximizing player is assured of and the minimum score that the minimizing player is assured of respectively. Initially alpha is negative infinity and beta is positive infinity, i.e. both players start with their lowest possible score. It can happen that when choosing a certain branch of a certain node the minimum score that the minimizing player is assured of becomes less than the maximum score that the maximizing player is assured of (beta <= alpha). If this is the case, the parent node should not choose this node, because it will make the score for the parent node worse. Therefore, the other branches of the node do not have to be explored.
  + - 1. **Explain how game of chess may benefit from min-max and alpha-beta pruning algorithm.**
* A lot of computation time of chess computers is spent in evaluating leafs of the game tree.
* Time is wasted on bad positions.
* In this research, a method that predicts the maximum evaluation result of sibling chess positions is defined. The idea is to prune brothers of a bad position without information loss.
* The resulting algorithm is a forward pruning method on leafs of the game tree, which gives correct minimax results.
* A maximum positional difference of the evaluation function on siblings must be correctly measured or assessed for the algorithm to work properly.
* The results of this thesis cannot be generalized because of the dependencies on the evaluation function, but are intended as a proof of concept and show that it is worthwhile to investigate Sibling Prediction Pruning Alpha-Beta in a broader context.