Krisha Bhambani – 21332

ADS MINI PROJECT

TITLE : **TIC-TAC-TOE**

Problem Definition : Design and implement Tic-Tac-Toe game using concepts of advance data structures in C++.

Core functionalities and concepts used:

1. Trees
2. Minmax implementation of trees
3. Alpha beta pruning

ADS Concepts:

Trees.

- Tree is a hierarchical data structure which stores the information naturally in the form of hierarchy style.
- Tree is one of the most powerful and advanced data structures.
- It is a non-linear data structure compared to arrays, linked lists, stack and queue.

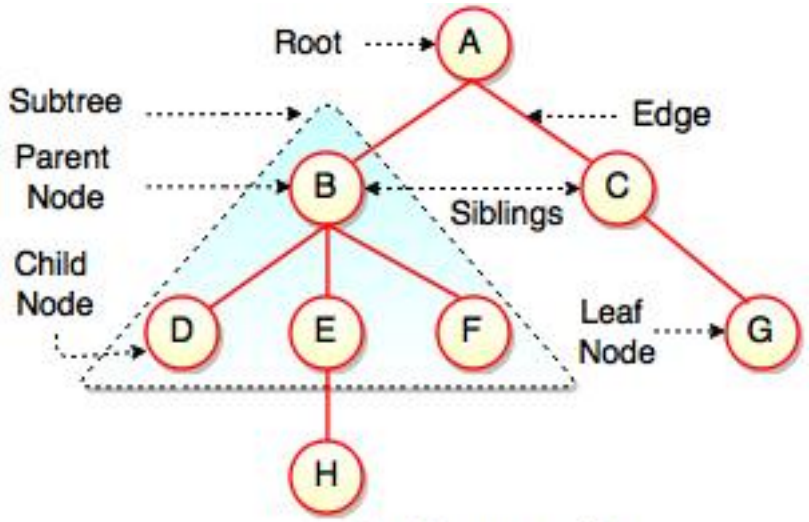

Fig. Structure of Tree

The above figure represents structure of a tree. Tree has 2 subtrees.
A is a parent of B and C.
B is called a child of A and also parent of D, E, F.

Tree is a collection of elements called Nodes, where each node can have arbitrary number of children.

Field	Description
Root	Root is a special node in a tree. The entire tree is referenced through it. It does not have a parent.
Parent Node	Parent node is an immediate predecessor of a node.
Child Node	All immediate successors of a node are its children.
Siblings	Nodes with the same parent are called Siblings.
Path	Path is a number of successive edges from source node to destination node.
Height of Node	Height of a node represents the number of edges on the longest path between that node and a leaf.
Height of Tree	Height of tree represents the height of its root node.
Depth of Node	Depth of a node represents the number of edges from the tree's root node to the node.

Degree of Node	Degree of a node represents a number of children of a node.
Edge	Edge is a connection between one node to another. It is a line between two nodes or a node and a leaf.

Mini-Max implementation of Trees

- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.
- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.
- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

Pseudo-code for MinMax Algorithm:

```
1. function minimax(node, depth, maximizingPlayer) is
2. if depth ==0 or node is a terminal node then
3. return static evaluation of node
4.
5. if MaximizingPlayer then      // for Maximizer Player
```

```
6. maxEva= -infinity
7.  for each child of node do
8.  eva= minimax(child, depth-1, false)
9. maxEva= max(maxEva,eva)      //gives Maximum of the values
10.       return maxEva
11.
12.       else                  // for Minimizer player
13.        minEva= +infinity
14.        for each child of node do
15.        eva= minimax(child, depth-1, true)
16.        minEva= min(minEva, eva)       //gives minimum of the valu
   es
17.        return minEva
```

Alpha Beta Pruning

- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
- As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning**. This involves two threshold parameter Alpha and beta for future expansion, so it is called **alpha-beta pruning**. It is also called as **Alpha-Beta Algorithm**.
- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.
- The two-parameter can be defined as:
    a. **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is **-∞**.

b. **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is **+∞**.

- The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

Condition for Alpha-beta pruning:

The main condition which required for alpha-beta pruning is:

1. α>=β

Key points about alpha-beta pruning:

- The Max player will only update the value of alpha.
- The Min player will only update the value of beta.
- While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.
- We will only pass the alpha, beta values to the child nodes.

Pseudo-code for Alpha-beta Pruning:

```
1. function minimax(node, depth, alpha, beta, maximizingPlayer) is
2. if depth ==0 or node is a terminal node then
3. return static evaluation of node
4.
5. if MaximizingPlayer then      // for Maximizer Player
6.    maxEva= -infinity
7.    for each child of node do
8.    eva= minimax(child, depth-1, alpha, beta, False)
9.   maxEva= max(maxEva, eva)
10.          alpha= max(alpha, maxEva)
11.          if beta<=alpha
12.         break
13.         return maxEva
14.
15.        else                  // for Minimizer player
16.          minEva= +infinity
17.          for each child of node do
```

```
18.         eva= minimax(child, depth-1, alpha, beta, true)
19.         minEva= min(minEva, eva)
20.         beta= min(beta, eva)
21.          if beta<=alpha
22.         break
23.        return minEva
```

About the project:

The project created is used for the implementation of Tic-tac-toe game . Its uses a class Game which creates the board and accepts X/O , its row and column positions. The class also contains a display function which is used to display the board. The above project also implements concepts like trees and Minmax implementation of trees. In addition it uses alpha beta pruning for optimizing the minmax algorithm .The game results in a tie if X and O are both out of moves.