Assignment 2
Simulation of adversarial search algorithms

The objective of this project is to design a graphical interface to simulate the execution of adversarial search algorithms, including depth-limited MINIMAX and MINIMAX Alpha-Beta Pruning algorithms.

Step 1: Construction of the Game Tree

The first step involves creating a binary tree where each node has only two children: left child and right child. The tree should have a depth of 5 as shown in Figure 1.

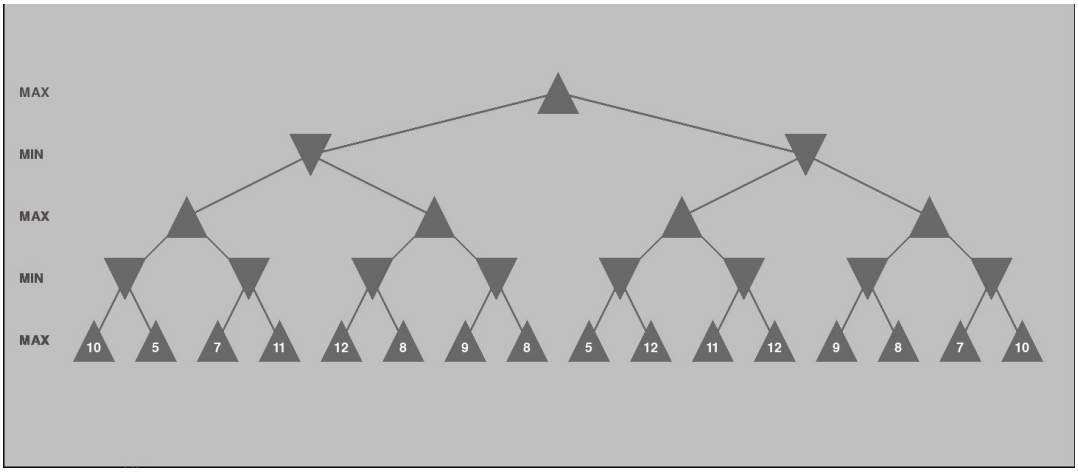


Figure 1 Initial Game Tree

To accomplish this, you need to define the following classes and functions:

The **Node** class, which represents a node in the game tree and contain the following attributes and functions:

- A link to the parent node, denoted as the *parent* variable.
- Links to the two child nodes, denoted as the *leftChild* and *rightChild* variables.
- The value of the node, denoted as the *value* attribute. This attribute is initially empty except for leaf nodes.
- The position of the node in the graphical interface, denoted as the *position* variable.
- A link to the child node from which the value is obtained during the application of the adversarial search algorithm, denoted as the *path* variable.
- The *init* function
- The *display* function that draws the node as a triangle.

The **Tree** class, which represents the game tree and contains the following functions:

- ✧ The *init* function that creates the root node
- ✧ The *createEmptyTree* recursive function that creates an empty tree of depth 5. The values of the leaf nodes are given by the following list: [10, 5, 7, 11, 12, 8, 9, 8, 5, 12, 11, 12, 9, 8, 7, 10]
- ✧ The *drawTree* recursive function that displays the empty game tree

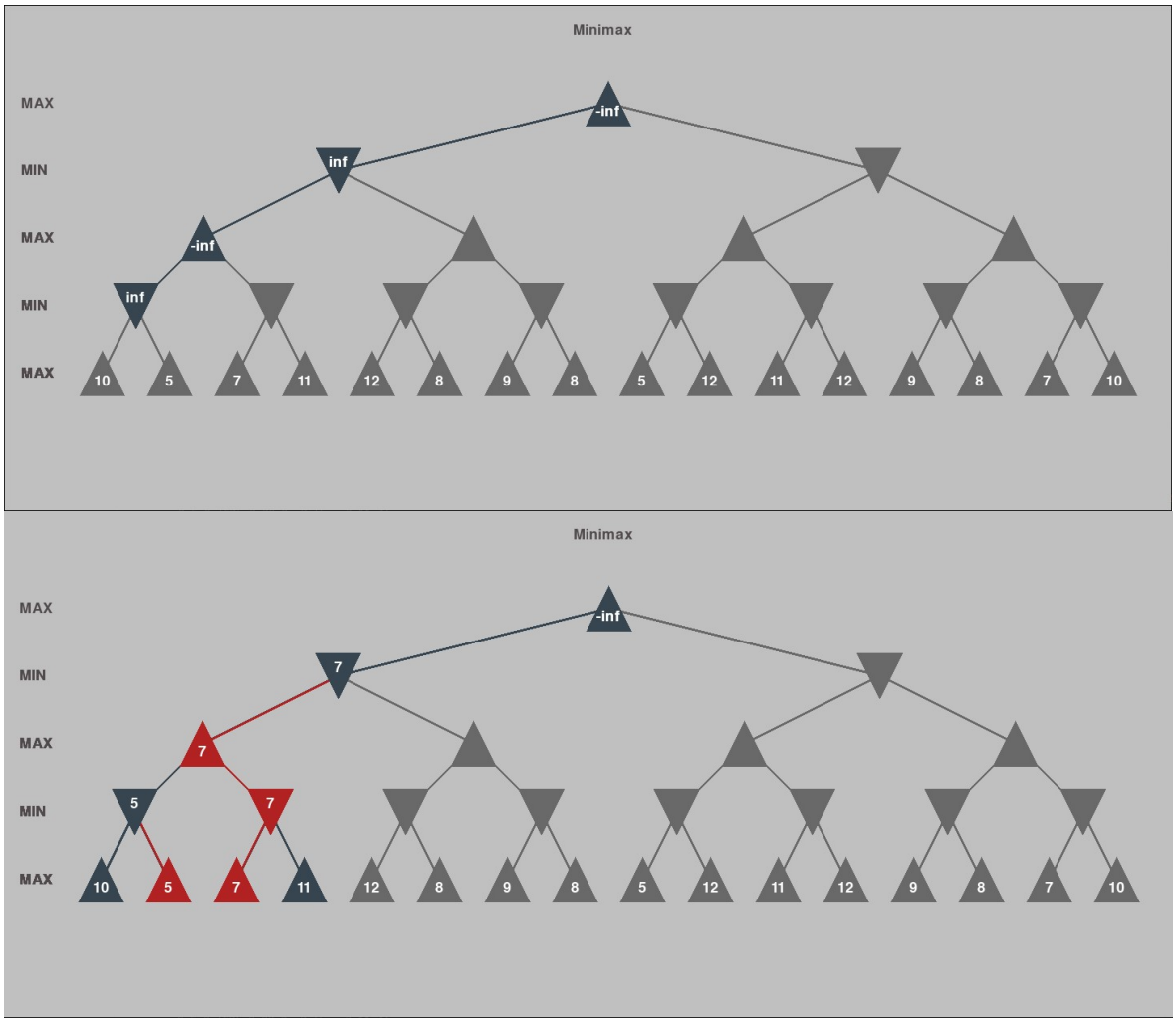
Step 2: Implementation of the MINIMAX algorithm

The second step involves implementing a recursive function that applies the MINIMAX algorithm to the tree defined in step 1 while simultaneously displaying the algorithm's execution in the graphical interface. This function is defined by the algorithm shown in Figure 2. A simulation of the MINIMAX function is depicted in Figure 3.

```
MINIMAX (node, depth, player=MAX) // Initial depth is 4
Begin
  If (depth == 0)
  {
     // Display the current node's value and mark it as explored
  }
  Else
  {
    // Mark the current node as explored
    If (player == MAX) // MAX = +1
    {
          node.value = -inf
          node.path = None
          child = node.leftChild
           // Mark the link between the current node and the left child node as explored
          MINIMAX (child, depth-1, -player)  // Apply the MINIMAX function on the left child
          If (child.value > node.value)
          {
                node.value = child.value
                node.path = child
                //  Make the necessary graphical updates
          }
          child = node.rightChild
           // Mark the link between the current node and the right child node as explored
          MINIMAX (child, depth-1, -player)  // Apply the MINIMAX function on the right child
          If (child.value > node.value)
          {
                node.value = child.value
                node.path = child
                //  Make the necessary graphical updates
          }
     }
     Else  // If the player is MIN (MIN = -1)
     {
          node.value = +inf
          node.path = None
          child = node.leftChild
          // Mark the link between the current node and the left child node as explored
          MINIMAX (child, depth-1, -player)  // Apply the MINIMAX function on the left child
          If (child.value < node.value)
```

```
        {
                node.value = child.value
                node.path = child
                //  Make the necessary graphical updates
        }
        child = node.rightChild
        // Mark the link between the current node and the right child node as explored
        MINIMAX (child, depth-1, -player)  // Apply the MINIMAX function on the right child
        If (child.value < node.value)
        {
                node.value = child.value
                node.path = child
                //  Make the necessary graphical updates
        }
    }
  }
End
```

Figure 2 MINIMAX algorithm



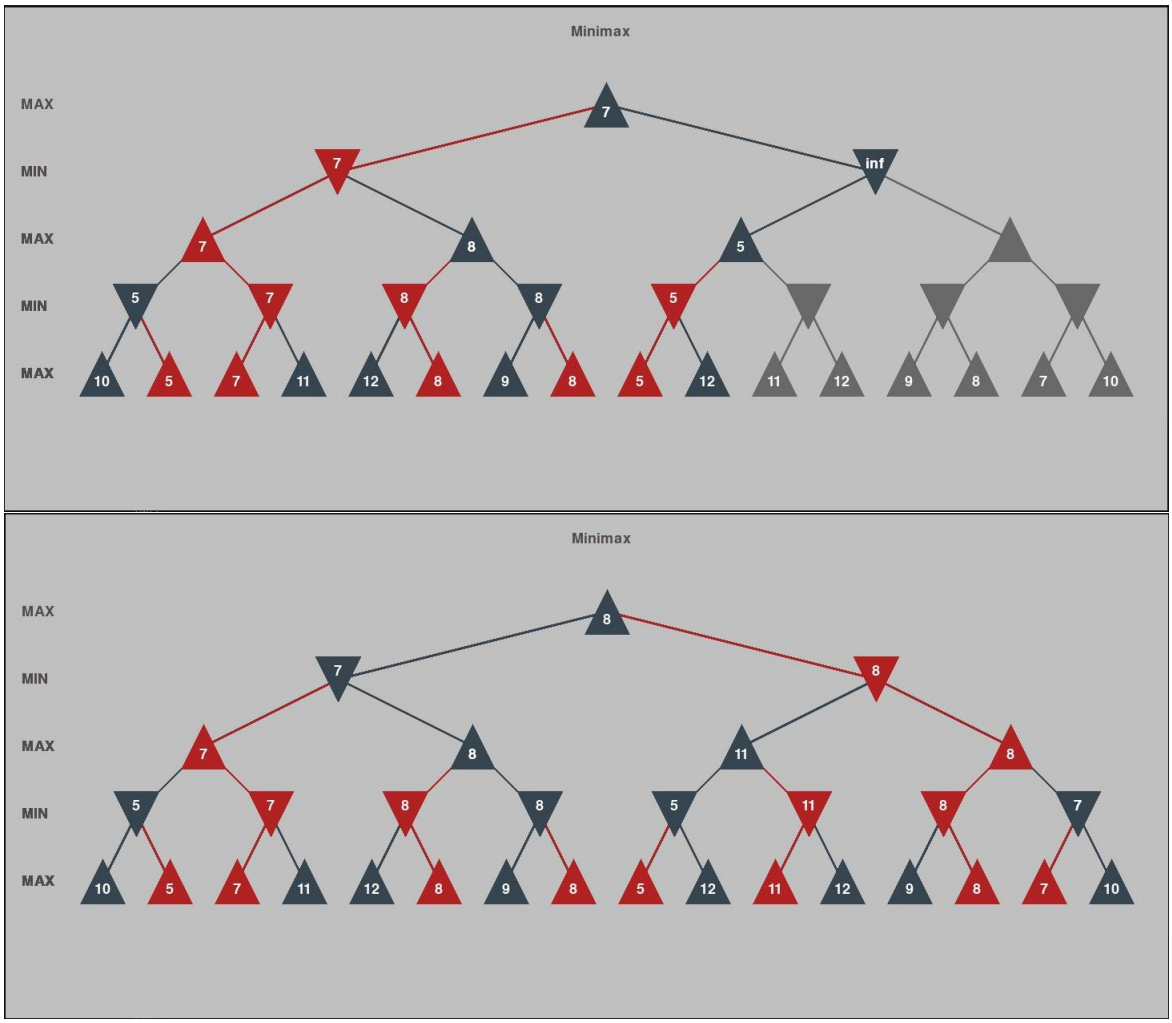


Figure 3 Execution of MINIMAX algorithm