21AIE111

Data Structures and Algorithms

BINARY TREE BASED

RACE GAME

TEAM – 10 MEMBERS:

KEERTHINATHAN -CH.EN.U4AIE21121
LAVANYA GOPINATH -CH.EN.U4AIE21166
SHEBA SULTHANA -CH.EN.U4AIE21148
VINEETH -CH.EN.U4AIE21162
VISHAL KRISHH -CH.EN.U4AIE21163

ACKNOWLEDGEMENT

We would like to acknowledge and give our warmest thanks to our mentor and guide R.Bhuvaneswari who made this work possible. Her guidance and advice carried us through all the stages of writing my project. Further, we would like to thank the various authors of the articles presented in the internet where we collected the details from.

We would also like to give special thanks to our parents for their continuous support and understanding when writing my project. Their prayer for us has always guided me in the right direction. We have taken efforts in this project. However, it would not have been possible without the kind support and help of my fellow classmates as well. We extend our sincere thanks for assisting in this project in all way they can.

Finally, we would like to thank God, for letting us through all the difficulties. We have experienced your guidance day by day. We will keep on trusting you for our future.

AIE DEPARTMENT

GROUP – 10

INDEX

INTRODUCTION

A data structure is a specialized format for organizing, processing, retrieving and storing data. There are several basic and advanced types of data structures, all designed to arrange data to suit a specific purpose. Data structures make it easy for users to access and work with the data they need in appropriate ways. Most importantly, data structures frame the organization of information so that machines and humans can better understand it.

In computer science and computer programming, a data structure may be selected or designed to store data for the purpose of using it with various algorithms. In some cases, the algorithm's basic operations are tightly coupled to the data structure's design. Each data structure contains information about the data values, relationships between the data and -- in some cases -- functions that can be applied to the data.

For instance, in an object-oriented programming language, the data structure and its associated methods are bound together as part of a class definition. In non-object-oriented languages, there may be functions defined to work with the data structure, but they are not technically part of the data structure.

Why are data structures important?

Typical base data types, such as integers or floating-point values, that are available in most computer programming languages are generally insufficient to capture the logical intent for data processing and use. Yet applications that ingest, manipulate and produce information must understand how data should be organized to simplify processing. Data structures bring together the data elements in a logical way and facilitate the effective use, persistence and sharing of data. They provide a formal model that describes the way the data elements are organized.

Data structures are the building blocks for more sophisticated applications. They are designed by composing data elements into a logical unit representing an abstract data type that has relevance to the algorithm or application. An example of an abstract data type is a "customer name" that is composed of the character strings for "first name," "middle name" and "last name."

It is not only important to use data structures, but it is also important to choose the proper data structure for each task. Choosing an ill-suited data structure could result in slow runtimes or unresponsive code

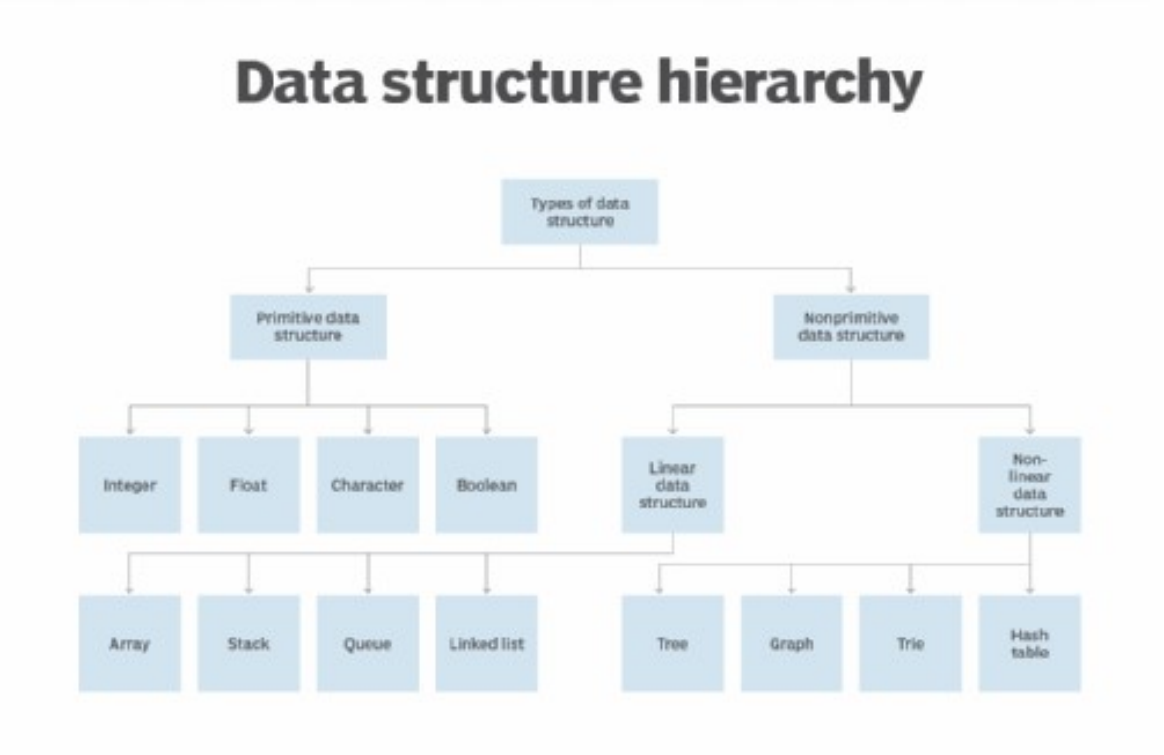
How data structures are used include the following:

- **Storing data.** Data structures are used for efficient data persistence, such as specifying the collection of attributes and corresponding structures used to store records in a database management system.
- **Managing resources and services.** Core operating system (OS) resources and services are enabled through the use of data structures such as linked lists for memory allocation, file directory management and file structure trees, as well as process scheduling queues.
- **Data exchange.** Data structures define the organization of information shared between applications, such as TCP/IP packets.
- **Ordering and sorting.** Data structures such as binary search trees -- also known as an ordered or sorted binary tree -- provide efficient methods of sorting objects, such as character strings used as tags. With data structures such as priority queues, programmers can manage items organized according to a specific priority.
- **Indexing**. Even more sophisticated data structures such as B-trees are used to index objects, such as those stored in a database.
- **Searching.** Indexes created using binary search trees, B-trees or hash tables speed the ability to find a specific sought-after item.
- **Scalability.** Big data applications use data structures for allocating and managing data storage across distributed storage locations, ensuring scalability and performance. Certain big data programming environments -- such as Apache Spark -- provide data structures that mirror the underlying structure of database records to simplify querying.

Characteristics of data structures

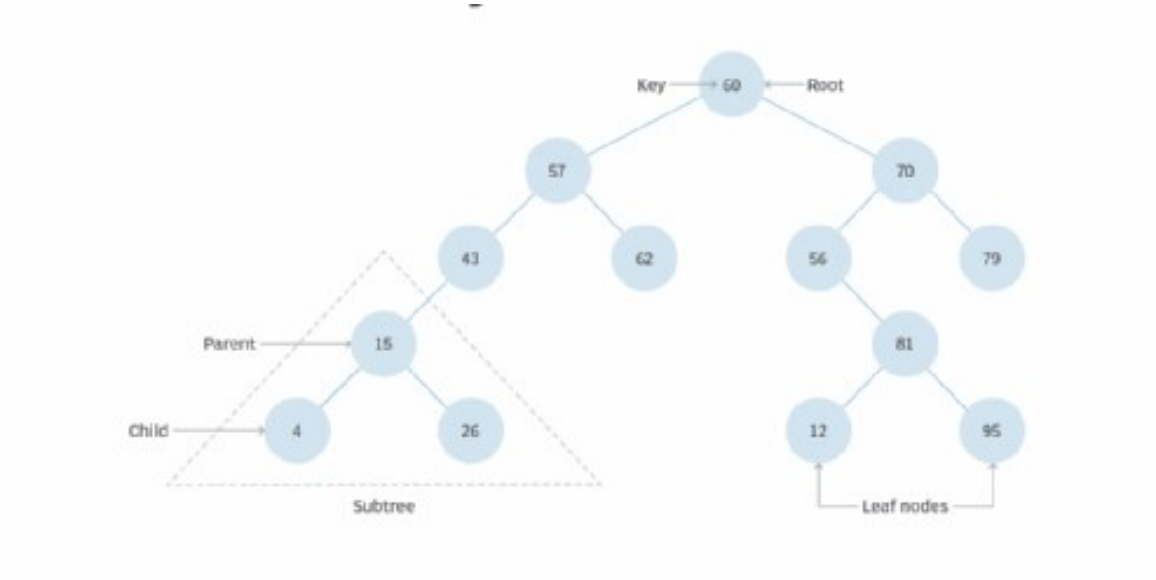
Data structures are often classified by their characteristics. The following three characteristics are examples:

1. **Linear or non-linear.** This characteristic describes whether the data items are arranged in sequential order, such as with an array, or in an unordered sequence, such as with a graph.
2. **Homogeneous or heterogeneous.** This characteristic describes whether all data items in a given repository are of the same type. One example is a collection of elements in an array, or of various types, such as an abstract data type defined as a structure in C or a class specification in Java.
3. **Static or dynamic.** This characteristic describes how the data structures are compiled. Static data structures have fixed sizes, structures and memory locations at compile time. Dynamic data structures have sizes, structures and memory locations that can shrink or expand, depending on the use.



TREE CONCEPT

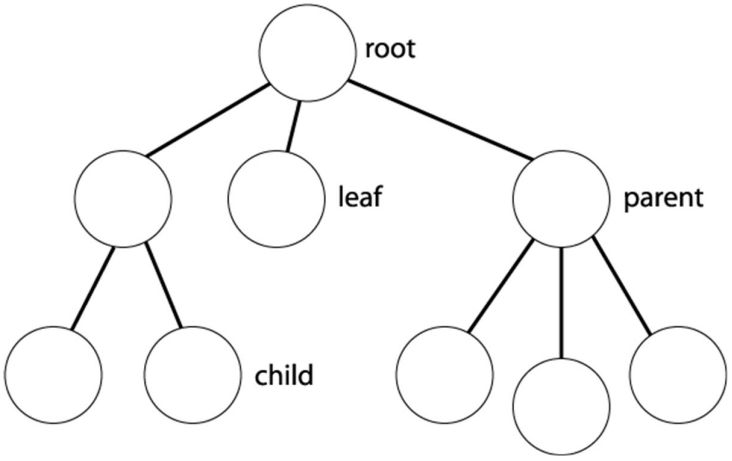
A *Tree* is a non-linear data structure where data objects are organized in terms of hierarchical relationship. The structure is non-linear in the sense that, unlike simple array and linked list implementation, data in a tree is not organized linearly. Each data element is stored in a structure called a *node*. The topmost or starting node of the (inverted) tree is called the *root node*. All nodes are linked with an edge and form hierarchical sub trees beginning with the root node. Tree data structure is useful on occasions where linear representation of data do not suffice, such as creating a family tree. Java provides two in-built classes, *TreeSet* and *TreeMap*, in Java Collection Framework that cater to the needs of the programmer to describe data elements in the aforesaid form.



What is java tree data structure?

Java tree classes are actual implementations of a specific variety of the tree data structure.

A tree, T, by definition, is a non-empty set of elements where one of these elements is called the root and the remaining elements (which may or may not be present in case of an empty tree rhyming with empty set of Set Theory) are partitioned further into sub trees of T.



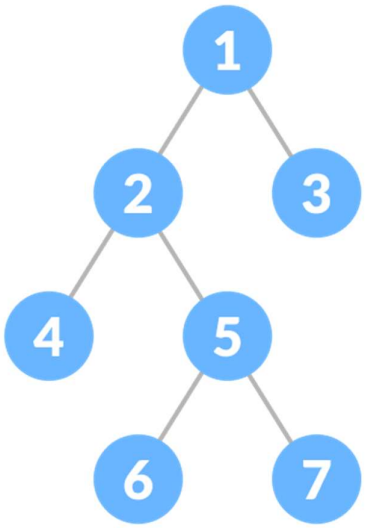
Binary trees have a few interesting properties when they're perfect:

- Property 1: The number of total nodes on each "level" doubles as you move down the tree.
- Property 2: The number of nodes on the last level is equal to the sum of the number of nodes on all other levels, plus 1

TYPES OF BINARY TREES

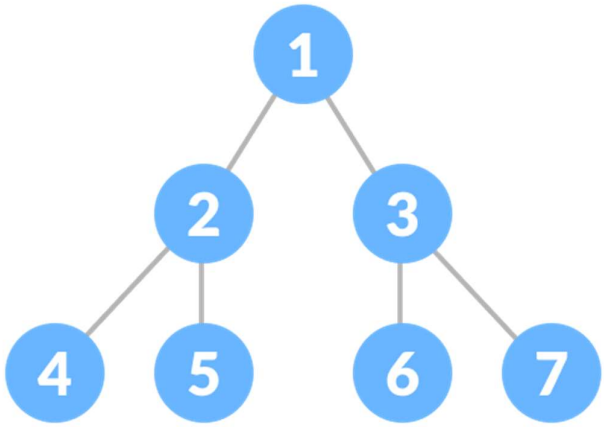
Full Binary Tree

A full binary tree is a binary tree where every node has exactly 0 or 2 children. The example of fully binary tress is:



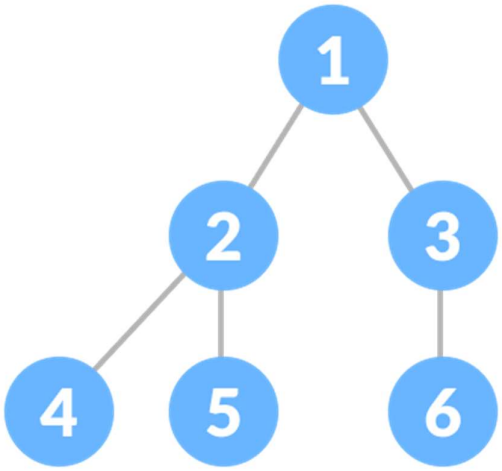
Perfect Binary Tree

A binary tree is perfect binary Tree if all internal nodes have two children and all leaves are at the same level. The example of perfect binary tress is:



Complete Binary Tree

A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible. An example of a complete binary tree is:



The following are the properties of the binary trees:

1. The minimum number of nodes at height h:

In any binary tree, the minimum number of nodes will be one more than the value of height. This is the number of nodes required to construct a tree.

Formula:

1. Height = h.
2. ***The minimum number of nodes = h+1.***
3. If h=3, then nodes will be 3+1= 4.

2. The maximum number of nodes at height h:

The maximum number of nodes that can be inserted in any binary tree of height h will be equal to $2^h - 1$.

Formula:

1. Height = h.
2. ***Maximum nodes to inserted = 2^h- 1.***
3. If h= 3, then 2^3- 1.
4. 8 -1= 7.

Therefore, the maximum number of nodes to be inserted for the height of h=3 will be 7.

3. Total number of leaf nodes:

The number of leaf nodes in a binary tree is equal to the nodes with degree two, plus one. Say a binary tree has two children. Then the total number of leaf nodes of that binary tree will be one greater than the nodes having two children.

Total number of leaf nodes = Nodes with 2 children + 1

Example:

Here, the total number of leaf nodes (no child or a successor) is 3, and the leaf nodes are E, F, and G.

Whereas the nodes with two children are 2. Node A and B have 2 children. Hence, this proves the property.

4. The maximum number of nodes at any level:

In a binary tree, the maximum number of nodes gained by any level is equal to the power of 2 for that level.

In a simpler words, the level is donated by n. Then, maximum nodes of that binary tree will be ***2^n***.

Example:

n = 2 then $2^2 = 4$.

Level 2 coverts the nodes (D, E, F, and G).

5. Minimum possible height or levels is equal to $Log_2(N+1)$:

This property says that the minimum number of levels or a height of a binary tree is the ***Log_2 of (N+1).***

Here N represents the maximum number of nodes possessed by a binary tree at the height h.

We have already discussed property 2 along with the example. Let's use that answer to calculate the height h.

1. N = 7 (Using Property#2)
2. Log_2 of (N+1) = Log_2 of (7+1)
3. $Log_2(8) = 3$

It is equal to the height chosen in the example of property 2.

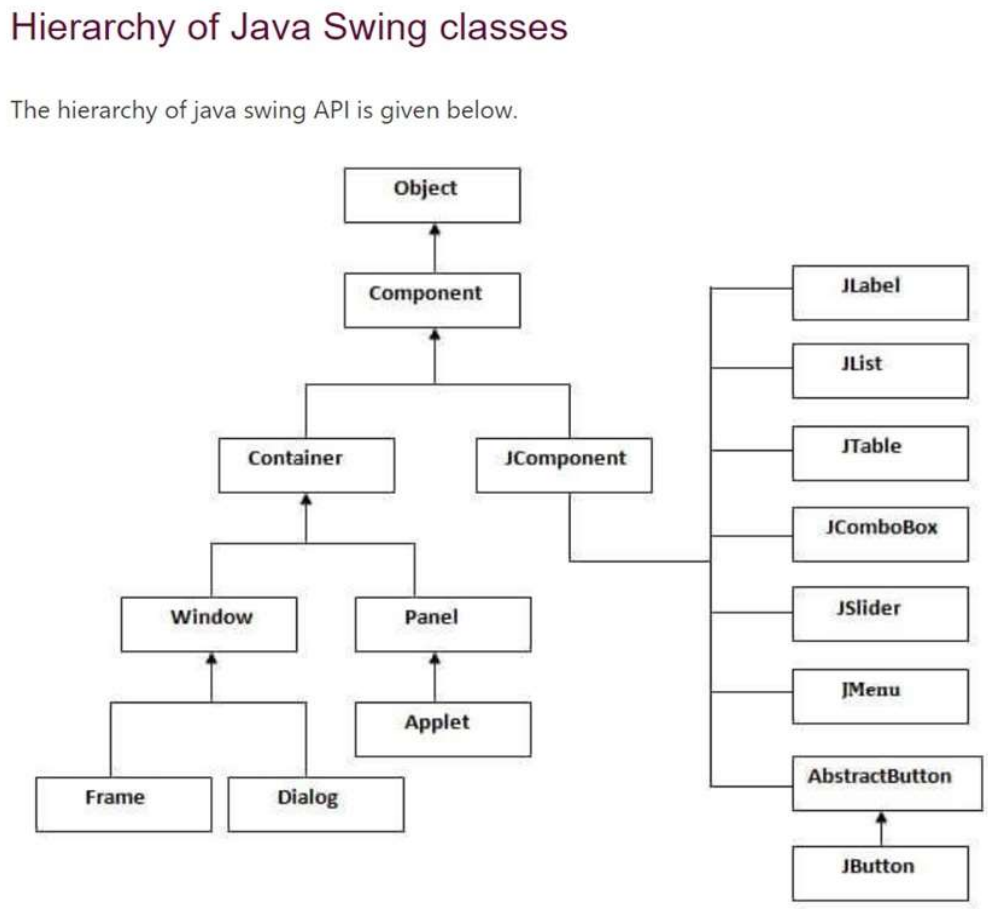
JAVA CODE IMPLEMENTATION

Our project is a simple race game based on binary tree done using GUI, where the player has to reach the end to win the game where the path to travel by each player will be randomized.

Java Swing

Java Swing is a lightweight Java graphical user interface (GUI) widget toolkit that includes a rich set of widgets. It is part of the Java Foundation Classes (JFC) and includes several packages for developing rich desktop applications in Java. Swing includes built-in controls such as trees, image buttons, tabbed panes, sliders, toolbars, color choosers, tables, and text areas to display HTTP or rich text format (RTF). Swing components are written entirely in Java and thus are platform-independent.

Swing offers customization of the look and feel of every component in an application without making significant changes to the application code. It also includes a pluggable look and feel feature, which allows it to emulate the appearance of native components while still having the advantage of platform independence. This particular feature makes writing applications in Swing easy and distinguishes it from other native programs.



Files imported for programming the project.

import java.awt.EventQueue → EventQueue is a platform-independent class that queues events, both from the underlying peer classes and from trusted application classes.

import javax.swing.JFrame → JFrame is a top-level container that provides a window on the screen. A frame is actually a base window on which other components rely, namely the menu bar, panels, labels, text fields, buttons, etc. Almost every other Swing application starts with the JFrame window.

import javax.swing.JToggleButton → A JToggleButton is a two-state button. The two states are selected and unselected. The JRadioButton and JCheckBox classes are subclasses of this class. When the user presses the toggle button, it toggles between being pressed or unpressed.

import javax.swing.JRadioButton→ JRadioButton class to create a radio button. Radio button is use to select one option from multiple options. It is used in filling forms, online objective papers and quiz.We add radio buttons in a ButtonGroup so that we can select only one radio button at a time. We use "ButtonGroup" class to create a ButtonGroup and add radio button in a group.

import javax.swing.JLabel → The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. Creates a JLabel instance with no image and with an empty string for the title.

import javax.swing.JOptionPane → The JOptionPane class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box. These dialog boxes are used to display information or get input from the user.

import java.awt.Font → **Font** is a class that belongs to the **java.awt** package. It implements the Serializable interface. FontUIResource is the direct known subclass of the Java Font class.

import java.awt.Color → Creates a color in the specified ColorSpace with the color components specified in the float array .

import javax.swing.JButton→ The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed.

import java.awt.event.ActionListener → ActionListener in Java is a class that is responsible for handling all action events such as when the user clicks on a component. Mostly, action listeners are used for JButtons. An ActionListener can be used by the implements keyword to the class definition.

import java.awt.event.ActionEvent → Uses of ActionEvent in java. awt. Handles the actionPerformed event by invoking the actionPerformed methods on listener-a and listener-b.

import java.awt.Toolkit →Toolkit class is the abstract superclass of every implementation in the Abstract Window Toolkit. Subclasses of Toolkit are used to bind various components.

CODE

Game.java

```
import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JToggleButton;
import javax.swing.JRadioButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;

import java.awt.Font;
import javax.swing.SwingConstants;
import java.awt.Color;

import javax.swing.ButtonGroup;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.Toolkit;
import javax.swing.ImageIcon;

public class Game {
    //References
    private JFrame Race; // INITIALIZING THE JFRAME
    private NodeG dispNode; //
    JRadioButton leftRB1;// INITIALIZING THE JRADIOBUTTON (LEFT) FOR PLAYER-1
    JRadioButton rightRB1;// INITIALIZING THE JRADIOBUTTON (RIGJT) FOR PLAYER-1
    JRadioButton leftRB2;// INITIALIZING THE JRADIOBUTTON (LEFT) FOR PLAYER-2
    JRadioButton rightRB2;// INITIALIZING THE JRADIOBUTTON (RIGHT) FOR PLAYER-2
    JLabel lblforp1;// INITIALIZING THE JLABEL FOR PLAYER-1
    JLabel lblforp2;// INITIALIZING THE JLABEL FOR PLAYER - 2
```

```
    JLabel posShow1;// INITIALIZING THE JLABEL TO SHOW THE POSOTION OF PLAYER -1
    JLabel posShow2;// INITIALIZING THE JLABEL TO SHOW THE POSOTION OF PLAYER -1
    JButton moveButton;// INITIALIZING THE JBUTTON
    ButtonGroup g1;// INITIALIZING THE JBUTTON
    ButtonGroup g2;// INITIALIZING THE JBUTTON
    JLabel depth1;// INITIALIZING THE JLABEL TELLING THE DEPTH FOR PLAYER-1
    JLabel depth2;// INITIALIZING THE JLABEL TELLING THE DEPTH FOR PLAYER-2

    BinaryTree tree; // INITIALIZING THE BINARY TREE
    Node p1;//INITIALIZING THE NODE FOR PLAYER-1
    Node p2;//INITIALIZING THE NODE FOR PLAYER-1
    NodeG pd1;//INITIALIZING THE NODE FOR PLAYER-1
    NodeG pd2;//INITIALIZING THE NODE FOR PLAYER-2
    private JLabel message;//INITIALIZING JLABEL FOR DISPLAYING MESSAGE

    /**
     * Launch the application.
     */
    Run | Debug
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() { //IT RUNS UNDERLYING PEER CLASSES & TRUSTED APPLICATION CLASSES
            public void run() {
                try {
                    Game window = new Game();
                    window.Race.setVisible(b: true); // OPENING THE GUI WINDOW
                } catch (Exception e) {
                    e.printStackTrace(); // IT DIAGONES THE EXCEPTION
                }
            }
        });
```

```
    }

    /**
     * Create the application.
     */
    public Game() {
        // POPING UP THE FIRST WINDOW TO DISPLAY THE MESSAGE
        JOptionPane.showMessageDialog(parentComponent: null, message: "This is the two player game in which the player, who can go
                title: "Welcome",messageType: 1);

        // CALLING THE MAIN GAME WINDOW
        Race = new JFrame();
        Race.setTitle(title: "Binary Tree Race Game By Team 10 "); // SETTING THE MAIN GAME WINDOW NAME

        tree = new BinaryTree();
        BinaryTreeGui BTree = new BinaryTreeGui(Race); // CALLING THE BinaryTreeGui.java (CLASS)
        dispNode = new NodeG();
        dispNode = BTree.createTree(dispNode, depth: 0, data: 1, x: 800, y: 150); // ALLIGNING THE STRUCTURE OF FIRST NODE FOR TREE
        dispNode.label.setVisible(aFlag: true);// IT SET VISIBLE TRUE - TO DISPLAY THE CONTENT
        initialize();
//      tree.traverse(tree.n, dispNode);
        p1 = tree.n;
        p2 = tree.n;
        pd1 = dispNode;
        pd2 = dispNode;
        display();


    }
```

```
/**
 * Initialize the contents of the frame.
 */

private void initialize() {
    Race.setBounds(x: 100, y: 100, width: 1277, height: 656); // SETTING THE POSITION AND THE SIZE OF THE BUTTON
    Race.setExtendedState(JFrame.MAXIMIZED_BOTH); // SETTING THE FRAME TO BE AT MAXIMIZED SIZE
    Race.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // SETTING TO CLOSE THE APPLICATION BY CLICKING THE CLOSE WINDOW
    Race.getContentPane().setLayout(mgr: null); // ADDING THE PROPERTIES TO THE FRAME

    leftRB1 = new JRadioButton(text: "LEFT"); // CREATING A JRADIOBUTTON
    leftRB1.setFont(new Font(name: "Tahoma", Font.PLAIN, size: 15));// SETTING FONT FOR THE CONTENT
    leftRB1.setBounds(x: 64, y: 239, width: 111, height: 23);// SETTING SIZE OF THE BUTTON
    Race.getContentPane().add(leftRB1);// ADDING THE PROPERTIES TO THE FRAME

    rightRB1 = new JRadioButton(text: "RIGHT");// CREATING A JRADIOBUTTON
    rightRB1.setFont(new Font(name: "Tahoma", Font.PLAIN, size: 15));// SETTING FONT FOR THE CONTENT
    rightRB1.setBounds(x: 64, y: 265, width: 111, height: 23);// SETTING SIZE OF THE BUTTON
    Race.getContentPane().add(rightRB1);// ADDING THE PROPERTIES TO THE FRAME

    lblforp1 = new JLabel(text: "Player 1 is at Node "); // SETTING THE LABEL FOR PLAYER -1
    lblforp1.setFont(new Font(name: "Tahoma", Font.PLAIN, size: 20));// SETTING FONT FOR THE CONTENT
    lblforp1.setBounds(x: 64, y: 203, width: 172, height: 23);// SETTING SIZE OF THE LABEL
    Race.getContentPane().add(lblforp1);// ADDING THE PROPERTIES TO THE FRAME

    posShow1 = new JLabel(text: "1");// SETTING LABEL FOR PLAYER - 1 IN THE GAME BOARD
    posShow1.setOpaque(isOpaque: true);// SETTING THE BOX AS OPAQUE
    posShow1.setBackground(Color.BLACK);// BACKGROUND COLOUR AS BLACK
    posShow1.setForeground(Color.WHITE);// FOREGROUND COLOUR AS WHITE
```

```
    posShow1.setForeground(Color.WHITE);// FOREGROUND COLOUR AS WHITE
    posShow1.setHorizontalAlignment(SwingConstants.CENTER);// SETTING THE LABEL TO BE IN CENTRE
    posShow1.setFont(new Font(name: "Tahoma", Font.PLAIN, size: 26));// SETTING THE FORNT
    posShow1.setBounds(x: 246, y: 201, width: 45, height: 23);// SETTING THE SIZE
    Race.getContentPane().add(posShow1);// ADDING THE PROPERTIES TO THE FRAME

    lblforp2 = new JLabel(text: "Player 2 is at Node ");// SETTING THE LABEL FOR PLAYER -2
    lblforp2.setFont(new Font(name: "Tahoma", Font.PLAIN, size: 20));// SETTING FONT FOR THE CONTENT
    lblforp2.setBounds(x: 64, y: 375, width: 172, height: 23);// SETTING SIZE OF THE LABEL
    Race.getContentPane().add(lblforp2);// ADDING THE PROPERTIES TO THE FRAME

    posShow2 = new JLabel(text: "1");// SETTING LABEL FOR PLAYER - 1 IN THE GAME BOARD
    posShow2.setOpaque(isOpaque: true);// SETTING THE BOX AS OPAQUE
    posShow2.setHorizontalAlignment(SwingConstants.CENTER);// SETTING THE LABEL TO BE IN CENTRE
    posShow2.setForeground(Color.WHITE);// FOREGROUND COLOUR AS WHITE
    posShow2.setFont(new Font(name: "Tahoma", Font.PLAIN, size: 26));// SETTING THE FORNT
    posShow2.setBackground(Color.blue);// BACKGROUND COLOUR AS BLUE
    posShow2.setBounds(x: 246, y: 373, width: 45, height: 23);// SETTING SIZE OF THE LABEL
    Race.getContentPane().add(posShow2);// ADDING THE PROPERTIES TO THE FRAME

    leftRB2 = new JRadioButton(text: "LEFT");// CREATING A JRADIOBUTTON
    leftRB2.setFont(new Font(name: "Tahoma", Font.PLAIN, size: 15));// SETTING FONT FOR THE CONTENT
    leftRB2.setBounds(x: 64, y: 411, width: 111, height: 23);// SETTING SIZE OF THE BUTTON
    Race.getContentPane().add(leftRB2);// ADDING THE PROPERTIES TO THE FRAME

    rightRB2 = new JRadioButton(text: "RIGHT");// CREATING A JRADIOBUTTON
    rightRB2.setFont(new Font(name: "Tahoma", Font.PLAIN, size: 15));// SETTING FONT FOR THE CONTENT
    rightRB2.setBounds(x: 64, y: 437, width: 111, height: 23);// SETTING SIZE OF THE BUTTON
    Race.getContentPane().add(rightRB2);// ADDING THE PROPERTIES TO THE FRAME
```

```
moveButton = new JButton(text: "Move");// CREATING A JRADIOBUTTON
moveButton.setFont(new Font(name: "Tahoma", Font.PLAIN, size: 27));// SETTING FONT FOR THE CONTENT
moveButton.setBounds(x: 64, y: 520, width: 227, height: 51);// SETTING SIZE OF THE BUTTON
Race.getContentPane().add(moveButton);// ADDING THE PROPERTIES TO THE FRAME

g1 = new ButtonGroup();
g1.add(leftRB1);// ADDING LEFT BUTTON OF PLAYER 1
g1.add(rightRB1);// ADDING RIGHT BUTTION FOR PLAYER 1

g2 = new ButtonGroup();
g2.add(leftRB2);// ADDING LEFT BUTTION FOR PLAYER 2
g2.add(rightRB2);// ADDING RIGHT BUTTION FOR PLAYER 2

JLabel lblNewLabel_1 = new JLabel(text: "depth : ");//SETTING THE LABLE
lblNewLabel_1.setFont(new Font(name: "Tahoma", Font.PLAIN, size: 15));// SETTING FONT FOR THE CONTENT
lblNewLabel_1.setBounds(x: 64, y: 297, width: 54, height: 23);// SETTING SIZE OF THE FONT
Race.getContentPane().add(lblNewLabel_1);// ADDING THE PROPERTIES TO THE FRAME

depth1 = new JLabel(text: "0");//SETTING THE LABLE
depth1.setFont(new Font(name: "Tahoma", Font.BOLD, size: 18));// SETTING FONT FOR THE CONTENT
depth1.setBounds(x: 128, y: 297, width: 54, height: 23);// SETTING SIZE OF THE FONT
Race.getContentPane().add(depth1);// ADDING THE PROPERTIES TO THE FRAME

JLabel lblNewLabel_1_1 = new JLabel(text: "depth : ");//SETTING THE LABLE
lblNewLabel_1_1.setFont(new Font(name: "Tahoma", Font.PLAIN, size: 15));// SETTING FONT FOR THE CONTENT
lblNewLabel_1_1.setBounds(x: 64, y: 467, width: 54, height: 23);// SETTING SIZE OF THE FONT
Race.getContentPane().add(lblNewLabel_1_1);// ADDING THE PROPERTIES TO THE FRAME
```

```
depth2 = new JLabel(text: "0");//SETTING THE LABLE
depth2.setFont(new Font(name: "Tahoma", Font.BOLD, size: 18));// SETTING FONT FOR THE CONTENT
depth2.setBounds(x: 128, y: 467, width: 54, height: 23);// SETTING SIZE OF THE FONT
Race.getContentPane().add(depth2);// ADDING THE PROPERTIES TO THE FRAME

message = new JLabel(text: "");//SETTING THE LABLE
message.setFont(new Font(name: "Tahoma", Font.PLAIN, size: 30));// SETTING FONT FOR THE CONTENT
message.setBounds(x: 64, y: 125, width: 430, height: 45);// SETTING SIZE OF THE FONT
Race.getContentPane().add(message);// ADDING THE PROPERTIES TO THE FRAME

JLabel lblNewLabel_2 = new JLabel(text: "");//SETTING THE LABLE
lblNewLabel_2.setBounds(x: 522, y: 28, width: 546, height: 106);// SETTING SIZE OF THE FONT
Race.getContentPane().add(lblNewLabel_2);// ADDING THE PROPERTIES TO THE FRAME

moveButton.addActionListener(new ActionListener() {// WORKING OF THE BUTTON
    public void actionPerformed(ActionEvent e) {
        // SETTING THE CONDITION FOR THE GUI TO WORK
        // BOTH PLAYER HAS TO SELECT THE PATHE TO TRAVEL WHEN GIVEN TO CHOOSE
        try {
            if(((leftRB1.isSelected() || rightRB1.isSelected()) &&
                    (leftRB2.isSelected() || rightRB2.isSelected())) ||
                    (!leftRB1.isVisible() && !rightRB1.isVisible()) ||
                    (!leftRB2.isVisible() && !rightRB2.isVisible()) ) {
                gameplay();
                display();


            }
```

```
                // IF BOTH PLAYERS DIDN'T CHOOSE OR EITHER ONE PLAYER DIDN'T CHOOSE IT THROWS EXCEPTION
                else {
                    throw new Exception();
                }
            }
            // WHEN IT THROWS EXCEPTION IT DISPLAYS THE ERROR MESSOGE
            catch(Exception e1) {
                JOptionPane.showMessageDialog(Race, message: "Both player should select the path!",
                        title: "Can't Move", JOptionPane.ERROR_MESSAGE);
            }
        }
    });
}
// TO CALCULATE THE DEPTH OF THE TREE BY INCREMENTING
public static void incDepth(JLabel l) {
    int d = Integer.parseInt(l.getText()); // CHARACTER TO INTEGER
    d++; // INCREMENT
    l.setText(Integer.toString(d));// INTEGER TO STING
}
public void gameplay() {
    //Conditions for moving player 1
    if(leftRB1.isSelected()) { // IF PLAYER 1 CLICKING LEFT BUTTON
        pd1 = pd1.left;
        p1 = p1.left;
        posShow1.setText(pd1.label.getText());
        incDepth(depth1);//INCREMENT THE DEPTH OF THE PLAYER 1
    }
    else if(rightRB1.isSelected()) {  // IF PLAYER 1 CLICKING RIGHT BUTTON
        pd1 = pd1.right;
        p1 = p1.right;
```

```
        p1 = p1.right;
        posShow1.setText(pd1.label.getText());
        incDepth(depth1);//INCREMENT THE DEPTH OF THE PLAYER 1
    }
    if(leftRB2.isSelected()) { // IF PLAYER 2 CLICKING LEFT BUTTON
        pd2 = pd2.left;
        p2 = p2.left;
        posShow2.setText(pd2.label.getText());
        incDepth(depth2); //INCREMENT THE DEPTH OF THE PLAYER 2
    }
    else if(rightRB2.isSelected()) { // IF PLAYER 2 CLICKING RIGHT BUTTON
        pd2 = pd2.right;
        p2 = p2.right;
        posShow2.setText(pd2.label.getText());
        incDepth(depth2);//INCREMENT THE DEPTH OF THE PLAYER 2
    }
    g1.clearSelection();//IT WILL CLEAR THE DEPTH AFTER EACH INCREMENT OF THE PLAYER 1
    g2.clearSelection();//IT WILL CLEAR THE DEPTH AFTER EACH INCREMENT OF THE PLAYER 2

}
public static boolean isGreaterThan(JLabel l1 , JLabel l2) {
    int c = Integer.parseInt(l1.getText());//
    int d = Integer.parseInt(l2.getText());//
    return c > d;
}
public void display(){ //TO DISPLAY

    if(pd1.label.getText().equals(pd2.label.getText())) {   //then pd1==pd2 points at some gui node
        pd1.label.setBackground(Color.red); //AT THE FIRST STEP OF THE GAME
        pd1.label.setForeground(Color.WHITE);
```

```
        }else {
            if(pd1.label.getText().equals(Integer.toString(p1.number)) ) {
                pd1.label.setOpaque(isOpaque: true);//IF PLAYER 1 CHANGES ITS POSITION IN TREE
                pd1.label.setBackground(Color.black);
                pd1.label.setForeground(Color.WHITE);
            }
            if(pd2.label.getText().equals(Integer.toString(p2.number))) {
                pd2.label.setBackground(Color.blue);//IF PLAYER 2 CHANGES ITS POSITION IN TREE
                pd2.label.setForeground(Color.WHITE);
            }
        }
        //Conditions to in display if there is way for p1
        if(p1.left != null) {
            pd1.left.label.setVisible(aFlag: true);//THE CHILDREN WILL DISPLAY
            leftRB1.setVisible(aFlag: true);
        }else {
            leftRB1.setVisible(aFlag: false);//IT WONT DISPLAYED THE CHILDREN
        }
        if(p1.right != null) {
            pd1.right.label.setVisible(aFlag: true);//THE CHILDREN WILL DISPLAY
            rightRB1.setVisible(aFlag: true);
        }else {
            rightRB1.setVisible(aFlag: false);//IT WONT DISPLAYED THE CHILDREN
        }
        //Conditions to in display if there is way for p2
        if(p2.left != null) {
            pd2.left.label.setVisible(aFlag: true);//THE CHILDREN WILL DISPLAY
            leftRB2.setVisible(aFlag: true);
```

```
        }else {
            leftRB2.setVisible(aFlag: false);//IT WONT DISPLAYED THE CHILDREN
        }
        if(p2.right != null) {
            pd2.right.label.setVisible(aFlag: true);//THE CHILDREN WILL DISPLAY
            rightRB2.setVisible(aFlag: true);
        }else {
            rightRB2.setVisible(aFlag: false);//IT WONT DISPLAYED THE CHILDREN
        }
        if(p2.left == null && p2.right == null &&
                p1.left == null && p1.right == null) {
                    //DEPTH 1 > DEPTH 2
            if(isGreaterThan(depth1,depth2)) {
                message.setText(text: "Player 1 won the Game!");
                JOptionPane.showMessageDialog(parentComponent: null, message: "Player 1 won the Game!",
                            title: "Thanks for Playing",messageType: 1);

            }
                //DEPTH 2 > DEPTH 1
            else if(isGreaterThan(depth2,depth1)) {
                message.setText(text: "Player 2 won the Game!");
                JOptionPane.showMessageDialog(parentComponent: null, message: "Player 2 won the Game!",
                            title: "Thanks for Playing",messageType: 1);

            }
            //AFTER END OF THE GAME
            else {
                JOptionPane.showMessageDialog(parentComponent: null, message: "DRAW Game!",
                            title: "Thanks for Playing",messageType: 1);
```

```
                message.setText(text: "Player 2 won the Game!");
                JOptionPane.showMessageDialog(parentComponent: null, message: "Player 2 won the Game!",
                        title: "Thanks for Playing",messageType: 1);

            }
            //AFTER END OF THE GAME
            else {
                JOptionPane.showMessageDialog(parentComponent: null, message: "DRAW Game!",
                        title: "Thanks for Playing",messageType: 1);

                message.setText(text: "Draw the Game!");
            }
            tree.traverse(tree.n, dispNode);

            //CLOSING POP UP WINDOW
            JOptionPane.showMessageDialog(parentComponent: null, message: "THANK YOU",
                    title: "HAVE A NICE DAY !!",messageType: 1);

        }
    }
}
```

BinaryTreeGui.java

```
import java.awt.Color;
import javax.swing.*;

import java.util.Random;

class NodeG{
    JLabel label;
    NodeG left;
    NodeG right;
}
public class BinaryTreeGui{
    static JFrame frame;
    static JLabel[] l = new JLabel[31];
    public BinaryTreeGui(JFrame frame) {
        this.frame = frame;
    }
    public static NodeG createTree(NodeG parent, int depth, int data, int x, int y) {
        if(depth != 5) { // MAXIMUM DEPTH OF THE TREE IS 5
            parent = new NodeG();
            l[data-1] =new JLabel(Integer.toString(data), JLabel.CENTER);//INTEGER TO STRING
            l[data-1].setSize(width: 20, height: 20);//SIZE OF THE BUTTON
            l[data-1].setLocation(x, y);//LOCATION OF THE BUTTON
            l[data-1].setOpaque(isOpaque: true);//LABEL WILL BE OPAQUE
            l[data-1].setBackground(Color.white);//LABEL BACKGROUND WILL BE WHITE
            l[data-1].setVisible(aFlag: false);
            frame.add(l[data-1]);//ADD THE CHILDREN AFTER EACH TURN OF THE BOTH PLAYERS
            parent.label = l[data-1] ;
            parent.left = createTree(parent.left,depth+1,data*2, x-10*(20/(depth*depth+1)), y+100);//CREATE THE TREE FROM TH
            parent.right = createTree(parent.right,depth+1,data*2+1,x+10*(20/ (depth*depth+1)),y+100);//CREATE THE TREE FROM
            return parent;
```

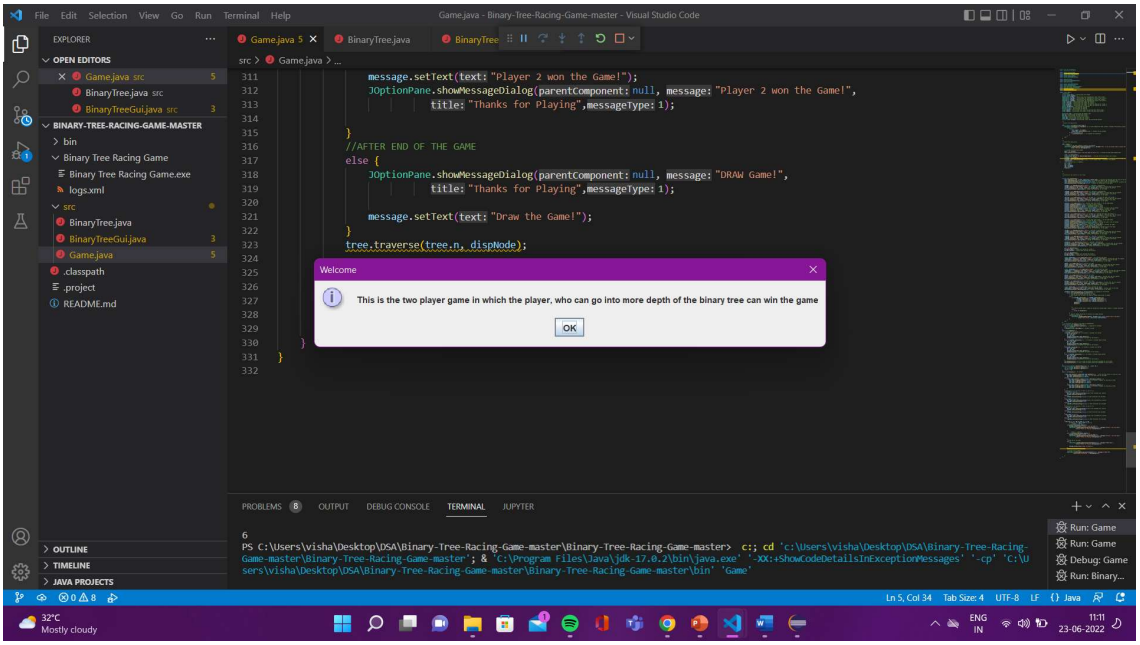
```
            return parent;
        }
        else {
            return null;
        }
    }
    Run | Debug
    public static void main(String[] args) {
        JFrame treeFrame = new JFrame();
        treeFrame.setBounds(x: 100, y: 100, width: 1277, height: 656);//SETTING UP THE TREE FRAME SIZE AND THE POSITION
        treeFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);//EXIT BUTTON WILL BE POP UP
        treeFrame.getContentPane().setLayout(mgr: null);

        BinaryTreeGui testTree = new BinaryTreeGui(treeFrame);
        NodeG dispNode = new NodeG();
        dispNode = createTree(dispNode, depth: 0, data: 1, x: 500, y: 10);//CREATE THE TREE IN THE POSITION
        frame.setVisible(b: true);//VISIBLITY OF THE TREE

    }
}
```

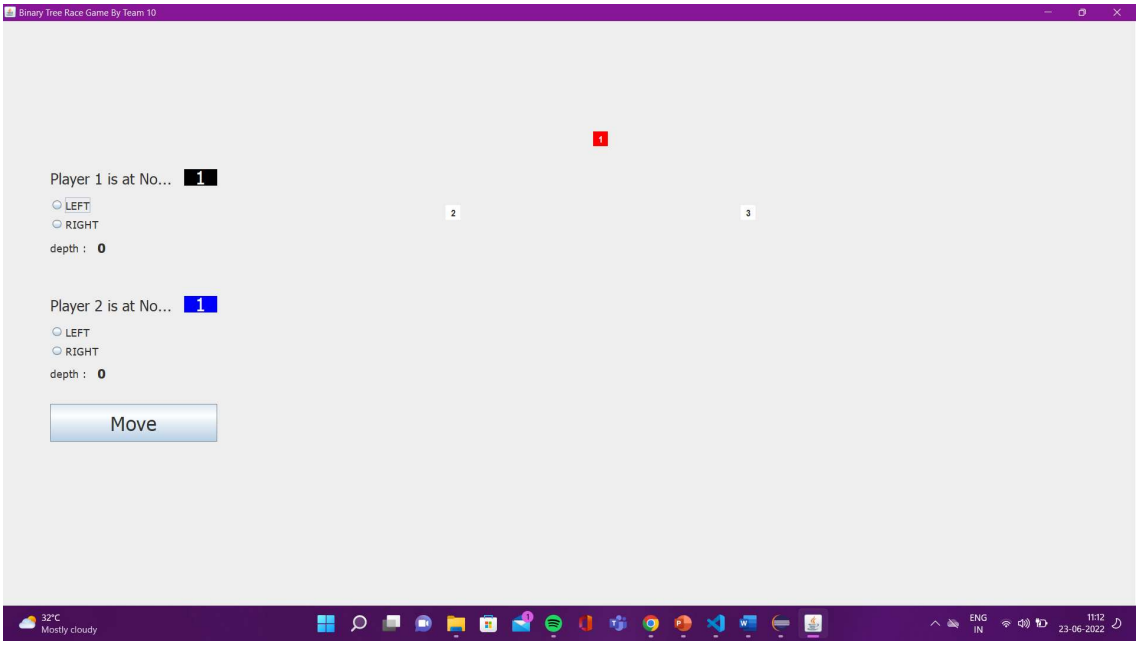
BinaryTree.java

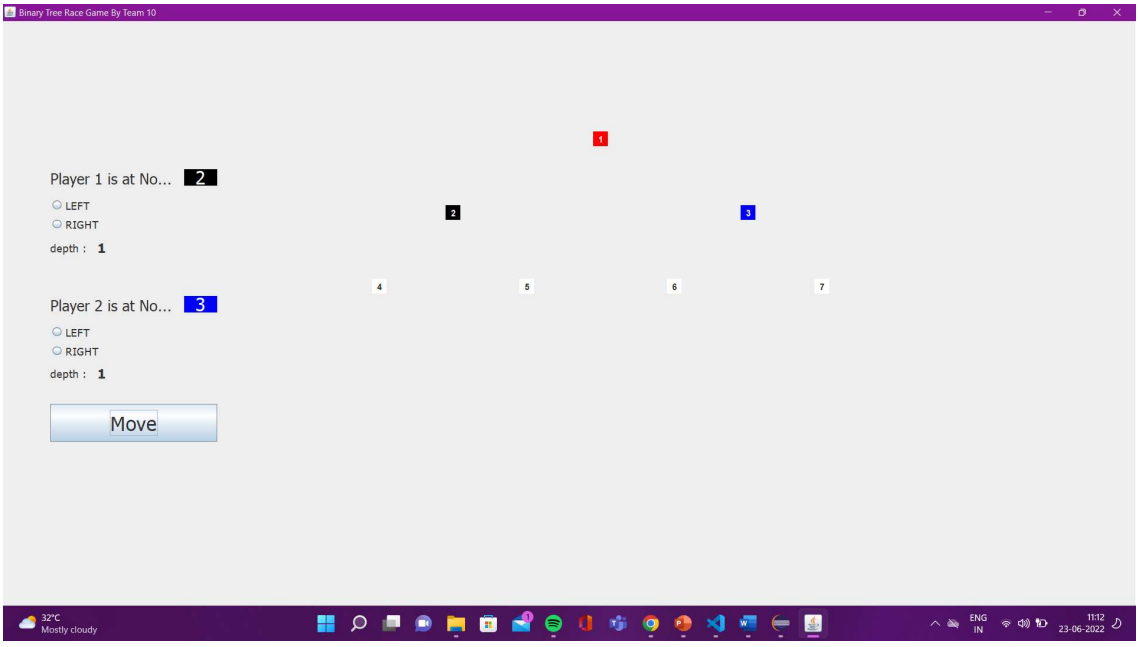
```
import java.util.Random;
class Node{
    int number;
    Node left;
    Node right;
}

public class BinaryTree {
    static Random r;
    Node n;
    BinaryTree(){
        r = new Random();
        n = createTree(n, depth: 0, data: 1, choice: 1);
    }
    public static Node createTree(Node parent, int depth, int data, int choice) {
        //If choice get 0 then no node else if 1 or 2 then make new node
        //This is done to increase the probability of creating node


        if(depth != 5) { //IF THE DEPTH IS NOT EQUALTO 5

            if(choice == 0)  { //when choice == 0 then don't create a node
                return null;
            }
            else {
                parent = new Node();
                parent.number = data;//if choice == 1 OR 2 then create new node 1/3 and 2/3 chances of no node or a node respectively.
                parent.left = createTree(parent.left,depth+1,data*2, r.nextInt(bound: 3));//TO CREATE THE LEFT NODE OF THE PARENT
                parent.right = createTree(parent.right,depth+1,data*2+1, r.nextInt(bound: 3));//TO CREATE THE RIGHT NODE OF THE PARENT
                return parent;
```

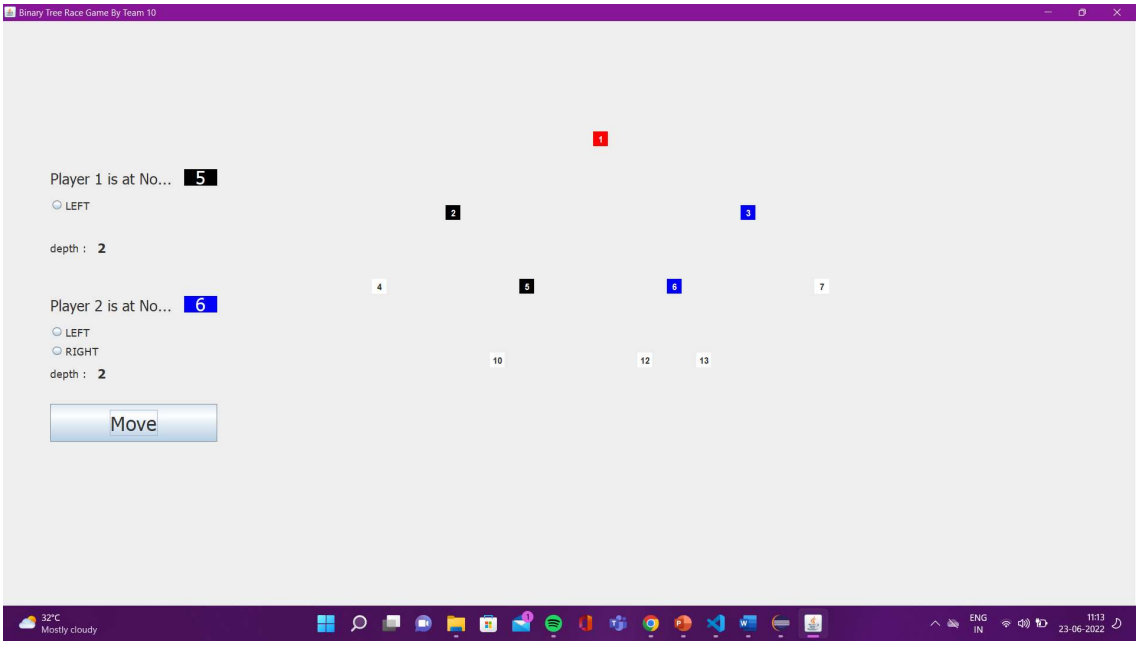
```
            }
        }
        else {
            return null;
        }
    }
    //TRAVERSE THE NODE OF THE TREE
    public static void traverse(Node parent, NodeG dispNode) {
        System.out.println(parent.number);
        dispNode.label.setVisible(aFlag: true);
        //LEFT NODE OF THE PARENT IS NOT EQUAL THEN IT WILL PRINT
        if(parent.left != null) {
            traverse(parent.left,dispNode.left);
        }
        //RIGHT NODE OF THE PARENT IS NOT EQUAL THEN IT WILL PRINT
        if(parent.right != null) {
            traverse(parent.right,dispNode.right);
        }
    }
}
```

OUTPUT

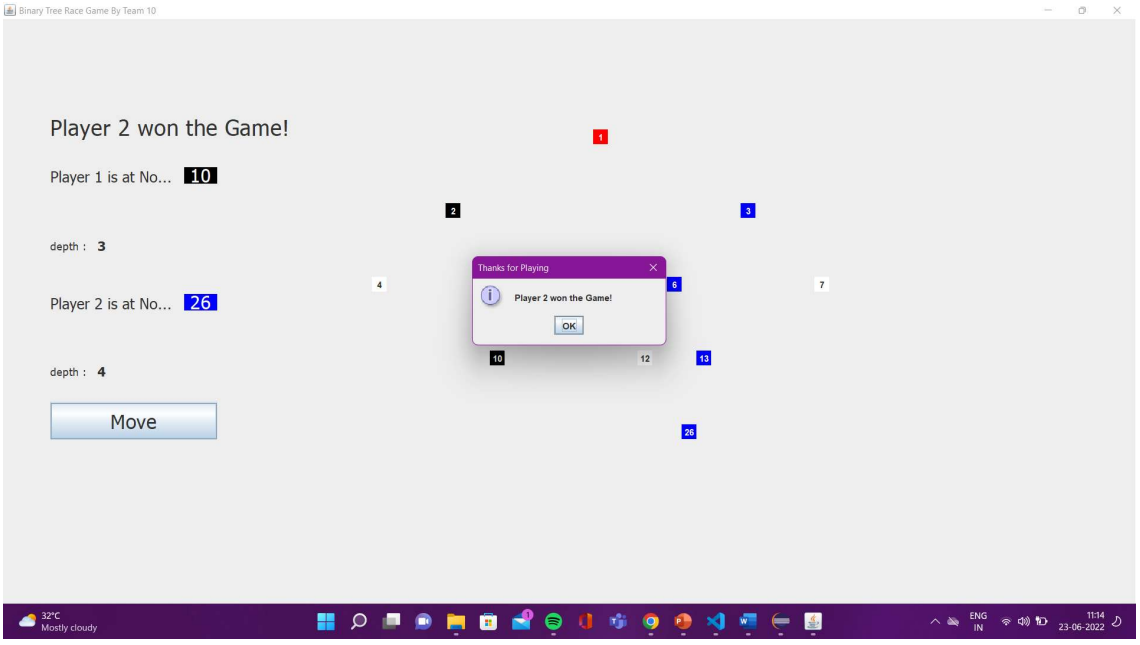



Binary Tree Race Game By Team 10
1
Player 1 is at No... 1
LEFT
RIGHT
depth : 0
2
3
Player 2 is at No... 1
LEFT
RIGHT
depth : 0
Move
32°C
Mostly cloudy
ENG
IN
11:12
23-06-2022

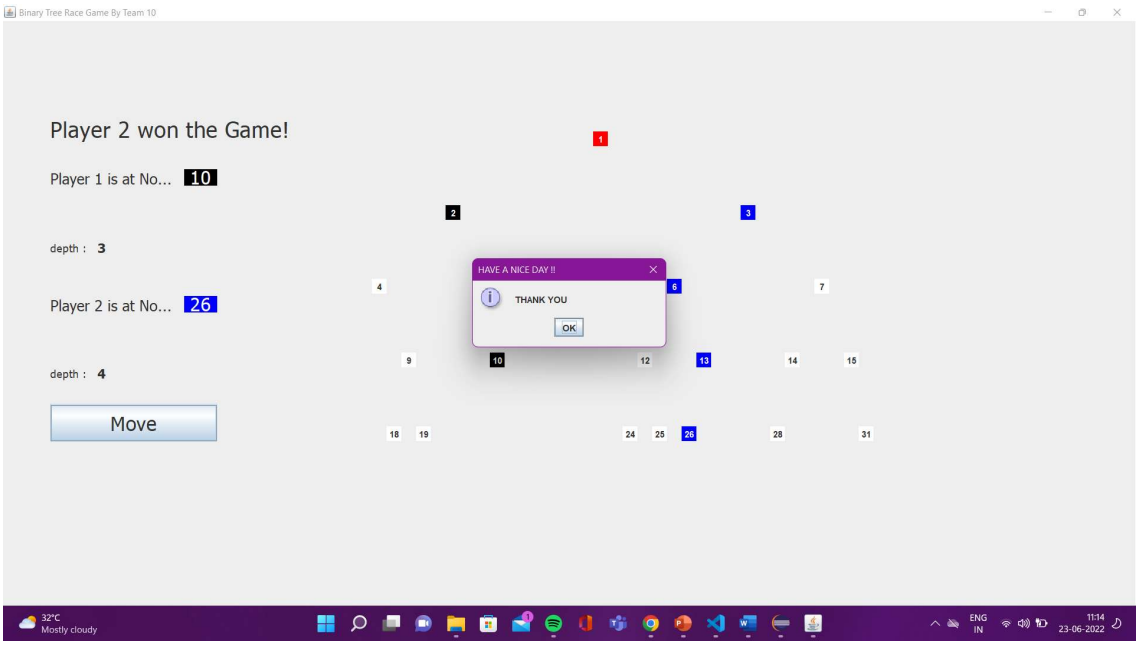

Binary Tree Race Game By Team 10
1
Player 1 is at No... 2
LEFT
RIGHT
depth : 1
2
3
4
5
6
7
Player 2 is at No... 3
LEFT
RIGHT
depth : 1
Move
32°C
Mostly cloudy
ENG
IN
11:12
23-06-2022



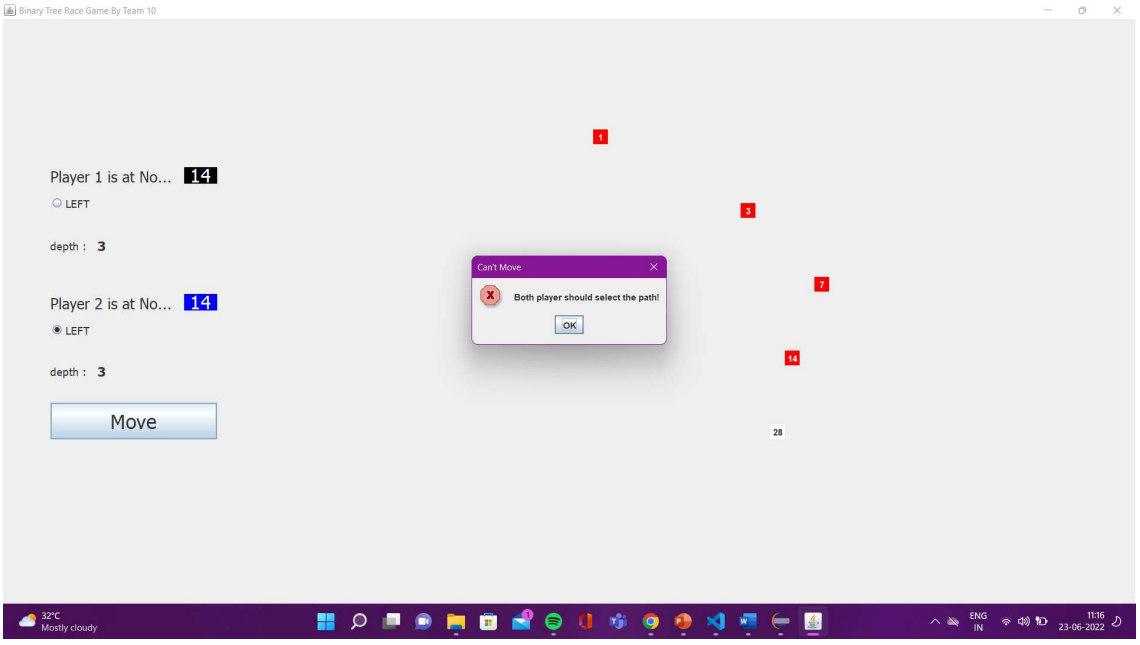
GAME WINNING:



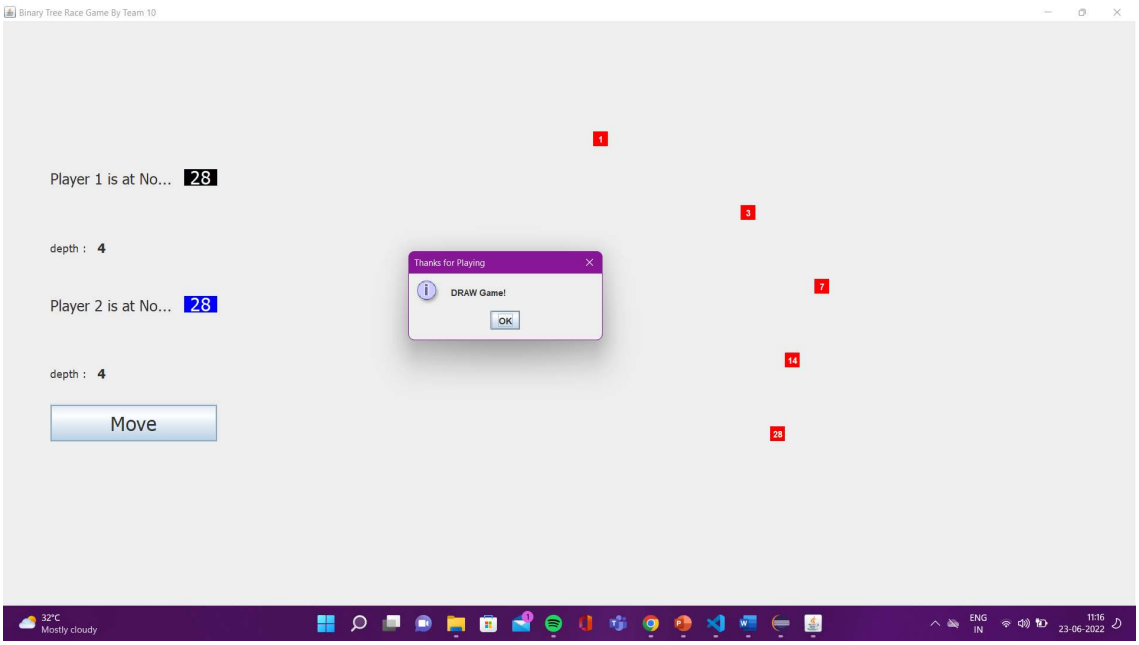
END


Binary Tree Race Game By Team 10
Player 2 won the Game!
Player 1 is at No... 10
depth : 3
Player 2 is at No... 26
depth : 4
Move
HAVE A NICE DAY !!
THANK YOU
OK
1
2
3
4
6
7
9
10
12
13
14
15
18
19
24
25
26
28
31
32°C
Mostly cloudy
ENG
IN
11:14
23-06-2022

ERROR


Binary Tree Race Game By Team 10
Player 1 is at No... 14
LEFT
depth : 3
Player 2 is at No... 14
LEFT
depth : 3
Move
Can't Move
Both player should select the path!
OK
1
3
7
14
28
32°C
Mostly cloudy
ENG
IN
11:16
23-06-2022

DRAW



CONCLUSION

Binary Tree is a special data structure used for data storage purposes. A binary tree has a special condition that each node can have a maximum of two children. A binary tree has the benefits of both an ordered array and a linked list as search is as quick as in a sorted array and insertion or deletion operation are as fast as in linked list.

From the project we have learnt about binary tree and its application. With the help of it we have created a race game by using random function and GUI.