OOP Assignment 5- Binary Tree

Problem Statement.

This problem will take the form of a yes/no guessing game, where the program will ask the user yes or no questions to try and guess what word the user is thinking of. This problem consists of storing a series of yes/no questions in the data structure of a binary tree. Each node will store a yes/no question as a string, and therefore each node will have a left and right child, constructing a proper binary tree. the tree will first be hardcoded in, starting off with 4 levels, and with each question, each yes answer points to the left child and each no answer points to the right child. We will be working with binaryNode objects, therefore we will have to cast each objects to Strings. This problem will also require us to traverse the tree and print out each question and prompt the user for an answer. We navigate through the tree based on the users answers, and once we reach a leaf node the tree will present the user with it's final guess. If the guess is wrong, we ask the user for the correct answer, and a new question to replace the current node with, and then replace the leaf node with the user's question, and set it's left child as the correct answer and it's right child as the incorrect answer that was the original leaf node. If the answer is correct, we will ask the user to chose to play again, quit the program, or save or load the tree. this means that I will have to construct 2 methods for saving the data from the current tree to an external file, and then loading data from an external file to populate the tree.

Analysis and design notes.

For this assignment, I will need to implement the binaryNode and binaryNodeinterface classes from canvas. I will start off by declaring a global binaryNode variable "root" which I will use as the root for the current tree I am building. In the main method, ill assign that root variable a string value, the root question, and pass it into the create tree method. The create tree method will take in the root as a parameter, and create the rest of the nodes for the tree. to do this, I will start off by creating the leaf nodes at the 4th layer of the tree, then I'll create the 3rd layer of nodes, and use the binaryNode constructor that takes the node data, left child and right child in as parameters, and do this repetedly up the tree, assigning each node with a previously created left and right child, until I reach the root node, which then I will use the setRightChild() and setLeftChild() methods to connect the root with the 2 nodes on the 2nd layer of the tree.

I will then have a method called display tree which will display all the data for the tree. this method will use the getHeight() and getNumberOfNodes() methods with the root node to get the tree height and total number of nodes, and will then call another method to print the data from the tree in the shape of a tree, to visualise all the nodes. I decided to name this method queueImplementation as I will use a queue object to arrange the nodes and store them to a string, to display to the user. The method will take the root of the tree as a parameter, and enqueue the root to the queue. I will then

have 2 variables, currentLevelnodes and nextLevelNodes that I will use to flag when the string will need a \n character to move to the next level of nodes. I will show basic pseudocode to show how this method will store the node data to a string in the form of the tree:

```
while (!queue.isEmpty()) {
currentNode = queue.dequeue();
string += currentNode.getData();
current levelNode --;

if(currentNode has a right child){
queue.enqueue(currentNode right child);
next levelNode++;
}

if(currentNode has a left child){
queue.enqueue(currentNode left child);
next levelNode++;
}

If (currentlevelNodes == 0) {
Treestring += \n;
Current levelnodes = next levelNodes;
Nextlevelnodes = 0;
}
}
Return string.
```

This way, the method will return a string, with each line containing all the nodes on that level.

I then need a method to traverse the tree and print the questions associated with each node to the user, scan in the user's response to the question, and work down through the tree depending on if the answer is yes/no, until it reaches a leaf node, then to print the final guess to the screen.
To do this, I will make a scanner object. Then I set a variable called currentNode to the root of the tree to start off, and run a loop that runs while(currentNode != leaf). In this loop, I will continuously print the data of the currentnode using the getData() method, and scan the user's input. If the user's input is yes, I set the cuurentNode = currentNode.getLeftChild(), and if the user's input is no, set currentNode to it's right child. This loop then continues until it reaches a leaf node.
Once it reaches a leaf node, it prints the leaf node data, the final guess, to the screen. It takes the user's input once again. If the user's input is yes, the method will give the user the option to play again, quit-(I will have a global variable Boolean called gameOn, and the traverse method will only run while(gameOn == true), if they decide to quit the game, gameOn will be set to false), save the current tree/game or load a previously saved tree/game. If the final guess is incorrect, and the user's input is no, it will prompt the user for the correct answer, store it as a string, prompt the user for a new question

to replace the current node that will be asked in it's place to differentiate the correct and incorrect answer, and store it as a string. Then the currentNode's question will be stored as a string, and passed into a new method along with the correct guess string, new question string, and currentNode object, this method will be called addquestion, and it will replace the current node and add a new question to the tree.
Addquestion() will create two new nodes with the correct guess string, and the incorrect guess string(currentNode's data), then it will replace currentNode's data with the new question string from the user's input, and set the correct guess node as the new currentNode's left child, and the incorrect guess node as the new cuurentNode's right child.

I will also have to construct methods to save and load the trees. I have had to change this part of my analysis notes, as I originally planned to store the tree's contents into a txt file and load it back in using bufferedReader and bufferedWriter objects, but have since changed the design to serialising and deserialising the tree in and out of a ser file.
To serialise the file, I will pass in the root of the tree the user wants to save, and the file path of the ser file as a string to the method. The method will create a fileoutputStream object leading to the ser file, and will call another serialise function that will be called recursively on itself with the parameters of the outputstream and the tree root, that will firstly write the contents of the root to the Fileoutput stream, and then recall the function with the node's left child, and then the object's right child. The recursive function will stop once the current node that was passed into the function is null, this is the base case.
To deserialise the function, I will use a FileInputstream object pointing to the same ser file, this function will work the same as the serialising function, where it will pass the Inputstream object into a function that will recursively read in the data from the nodes in the ser file and store them as a string, then creating a binary node object with that string, and recursively calling the function again by setting the leftChild of the newly created node as the next node in the file, calling the method again, and then setting the right child of the node as the next node in the file, recursively calling the method once again. This method's base case will stop once the data received from the input node is null.

Code: (implements arrayQueue and binaryNode)

```
import java.util.Scanner;
import java.io.*;
public class GuessingGame {
    private static BinaryNode<String> root; //root node
    private static boolean gameOn = true; //boolean to indicate if the user
wants to play the game or not
    public GuessingGame() {
        // initialize an empty tree
        root = null;
    }
```

```
    public static void saveTree() {
        try {
            //filename of the ser file we're storing the tree in
            serialize(root,
"/Users/ciarangray/BinaryTreeGuessingGame/tree.ser");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public static void loadGame() {
        try {
            //filename of the ser file we're loading the tree from
            root =
deserialize("/Users/ciarangray/BinaryTreeGuessingGame/tree.ser");
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
//method to populate the tree
    public static void createTree(BinaryNode<String> tree) {
        //pass in the root node created in main to this method
        //start with the leaf nodes
        BinaryNode<String> o = new BinaryNode<>("Jamie Vardy?");
        BinaryNode<String> n= new BinaryNode<>("Alexander Isac?");
        BinaryNode<String> m = new BinaryNode<>("Raheem Sterling?");
        BinaryNode<String> l = new BinaryNode<>("Erling Haaland?");
        BinaryNode<String> k = new BinaryNode<>("Matt le Tissier?");
        BinaryNode<String> j = new BinaryNode<>("alan shearer?");
        BinaryNode<String> i = new BinaryNode<>("robbie keane?");
        BinaryNode<String> h = new BinaryNode<>("roy keane?");
        //then use constructors to set the child nodes of the new nodes
        BinaryNode<String> g = new BinaryNode<>("is the team currently in
the premier league", n, o);
        BinaryNode<String> f = new BinaryNode<>("does he play for a
manchester team?", l, m);
        BinaryNode<String> e = new BinaryNode<>("is the team currently in
the premier league?", j, k);
        BinaryNode<String> d = new BinaryNode<>("did he play for a
manchester team?", h, i);
        BinaryNode<String> c = new BinaryNode<>("is he playing in a top 6
team?", f, g);
        BinaryNode<String> b = new BinaryNode<>("did he play in a top 6
team?", d, e);
        //set the left and right child of the root to complete the tree
        tree.setLeftChild(b);
        tree.setRightChild(c);
    }

    //method to add a new question to the tree
    public static void addQuestion(String question, String yesAnswer,
String noAnswer, BinaryNode<String> oldNode) {
        //create new nodes for the question and its answers
        BinaryNode<String> yesNode = new BinaryNode<>(yesAnswer);
        BinaryNode<String> noNode = new BinaryNode<>(noAnswer);
        //replace old node with the wrong guess with new question
        oldNode.setData(question);
        oldNode.setLeftChild(yesNode);
        oldNode.setRightChild(noNode);
    }
//method to serialize the tree and write it to a file
    public static void serialize(BinaryNode<String> root, String fileName)
```

```
throws IOException {
        try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(fileName))) {
            serialize2(root, out);
        }
    }
//this method then recursively serialize the binary tree
    private static void serialize2(BinaryNodeInterface<String> node,
ObjectOutputStream out) throws IOException {
       //base case to stop the recursion
        if (node == null) {
            out.writeObject(null);
            return;
        }
        //write the current node to the output stream
        out.writeObject(node.getData());
        //recursively call the left and right child
        serialize2(node.getLeftChild(), out);
        serialize2(node.getRightChild(), out);
    }
//metthod to deserialize the tree from the file
    public static BinaryNode<String> deserialize(String fileName) throws
IOException, ClassNotFoundException {
        try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream(fileName))) {
            // calls the helper method to perform the deserialisation and
cast the return value, the root, as a binary node object
            return (BinaryNode<String>) deserialize(in);
        }
    }
//this method recursively deserialises the tree and returns the root
    //method will only work if it returns a binaryNodeInterface object
    private static BinaryNodeInterface<String>
deserialize(ObjectInputStream in) throws IOException,
ClassNotFoundException {
        String data = (String) in.readObject(); //read data of current node
from input stream
        if (data == null) {
            return null;
        }
        //create new binaryNodeInterface object with the data read in from
the input stream
        BinaryNodeInterface<String> node = new BinaryNode<String>(data);
        //recursively deserialise the left and right children of the
current node
        node.setLeftChild(deserialize(in));
        node.setRightChild(deserialize(in));
        return node;
    }


    // Method to traverse the tree and ask the questions in each node
    public static void traverse() {
        //start with the rootNode
        BinaryNode<String> currentNode = root;
        Scanner scanner = new Scanner(System.in);
            //traverse the tree until reaching a leaf node, then ask the
final guess
            while (!currentNode.isLeaf()) {
                //print the current question
                System.out.println(currentNode.getData());
```

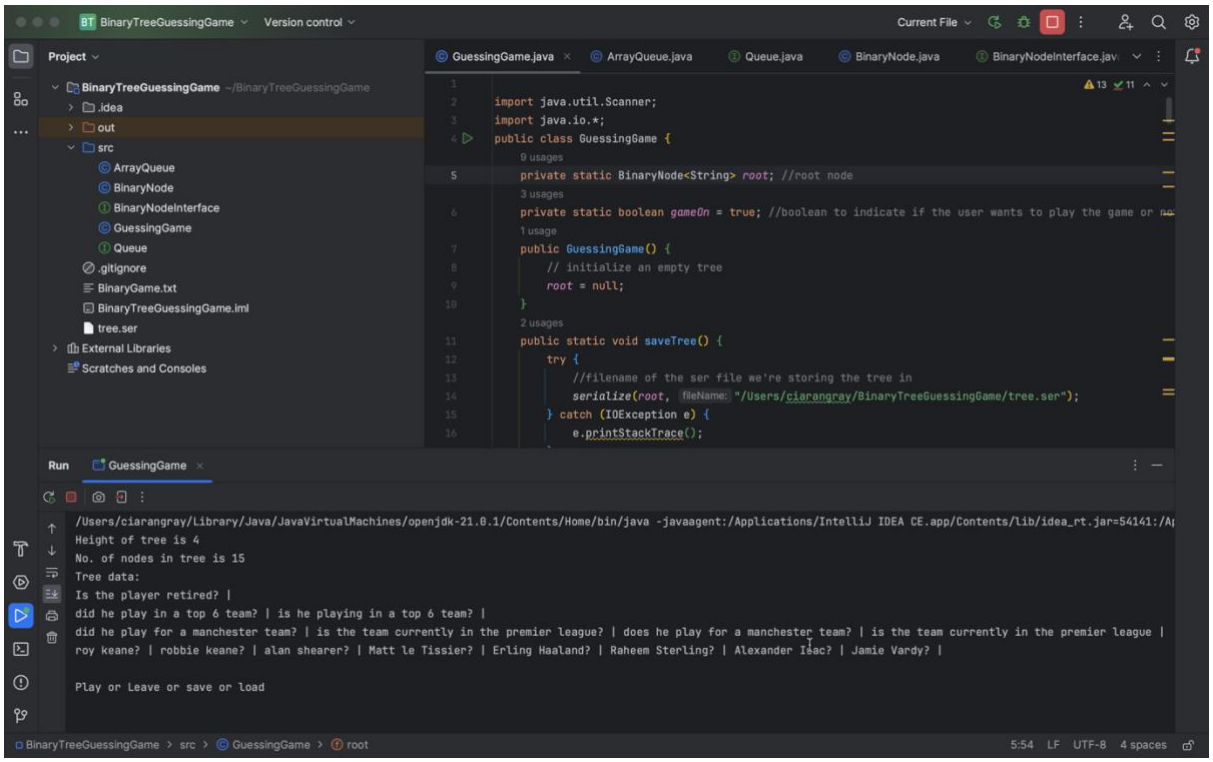
```
                //get yes/no answer from user
                String answer = scanner.nextLine().toLowerCase();
                //yes leads to left child
                if (answer.equals("yes")) {
                    currentNode = (BinaryNode<String>)
currentNode.getLeftChild();
                    //no leads to right child
                } else if (answer.equals("no")) {
                    currentNode = (BinaryNode<String>)
currentNode.getRightChild();
                } else {
                    System.out.println("Please answer 'yes' or 'no'.");
                }
            }

            //once we read a leaf node, print final guess
            System.out.println("Is it " + currentNode.getData() + "?");
        String guess = scanner.nextLine().toLowerCase();
            if (guess.equals("yes")) {
                System.out.println("I got it!");
            } else if (guess.equals("no")) {
                System.out.println("What was the answer?");
                //get correct answer
                String corrAns = scanner.nextLine();
                System.out.println("Give me another question i should of
asked about the answer being: " + corrAns);
                //get new question to replace the current node
                String newQ = scanner.nextLine();
                //replace current node with a new node and both children
nodes
                addQuestion(newQ, corrAns, currentNode.getData(),
currentNode);
            } else {
                System.out.println("Please answer 'yes' or 'no'.");
            }
            boolean nextGame = false;
            while(!nextGame) {
                System.out.println("Play or Leave or save or load");
                String game = scanner.nextLine().toLowerCase();

                if (game.equals("play")) {
                    nextGame = true;
                }
                if (game.equals("leave")) {
                    gameOn = false;
                    nextGame = true;
                }
                //save the current tree to file
                if (game.equals("save")) {
                    saveTree();
                }
                //load a previously stored tree
                if (game.equals("load")) {
                    loadGame();
                }
            }
    }
    //method to display tree stats
    public static void displayStats() {
        System.out.println("Height of tree is " + root.getHeight());
```

```
        System.out.println("No. of nodes in tree is " +
root.getNumberOfNodes());
        System.out.println("Tree data: \n" + queueImplementation(root));
    }
    //method to display the contents of the tree in a tree structure, using
a queue object
    public static String queueImplementation(BinaryNode<String> tree) {
        if(tree == null) {
            return "Tree is empty";
        }
        //initialize an array queue object
        ArrayQueue q = new ArrayQueue();
        //enqueue the root first
        q.enqueue(tree);
        //string to store the tree contents
        String treeString = "";
        int currentLevelNodes = 1;//int to store the number of nodes on
current level, on first level there is only the root so it starts at 1
        int nextLevelNodes = 0;//int to store next level nodes, which lets
us know when to move to the next level
        while(!q.isEmpty()) {
            //dequeue the nodes from the queue and add them to the string
                BinaryNode<String> currentNode = (BinaryNode<String>)
q.dequeue();
                treeString += currentNode.getData() + " | ";
                //counter here lets us know when we've added all the nodes
on this level to the string
                currentLevelNodes--;
                //enqueue the child nodes of "current node" and update the
nextLevelNodes counter
                if (currentNode.hasLeftChild()) {
                    q.enqueue(currentNode.getLeftChild());
                    nextLevelNodes++;
                }
                if (currentNode.hasRightChild()) {
                    q.enqueue(currentNode.getRightChild());
                    nextLevelNodes++;
                }
            if (currentLevelNodes == 0) {
                treeString += "\n"; //move to a new line in the string for
the next level
                currentLevelNodes = nextLevelNodes; //moving to the next
level so update currentLevelNodes
                nextLevelNodes = 0; //reset nodes in the next level
            }
        }
        return treeString; //return the string
    }

    // Example usage
    public static void main(String[] args) {
        GuessingGame GuessingGame = new GuessingGame();

        //create the root for the tree
        BinaryNode<String> tree = new BinaryNode<>("Is the player
retired?");
        root = tree;
        //populate the rest of the tree with this method
        createTree(root);
        //display the tree
        displayStats();
```

```
        Scanner scanner = new Scanner(System.in);
        boolean menu = true;
        while (menu) {
            System.out.println("Play or Leave or save or load");
            String game = scanner.nextLine().toLowerCase();

            if (game.equals("play")) {
                menu = false;
            }
            if (game.equals("leave")) {
                menu = false;
                gameOn = false;
            }
            //save the current tree to file
            if (game.equals("save")) {
                saveTree();
            }
            //load a previously stored tree
            if (game.equals("load")) {
                loadGame();
            }
        }
        //once the game has started, traverse the tree
        while (gameOn) {
            traverse();
        }
    }
}
```

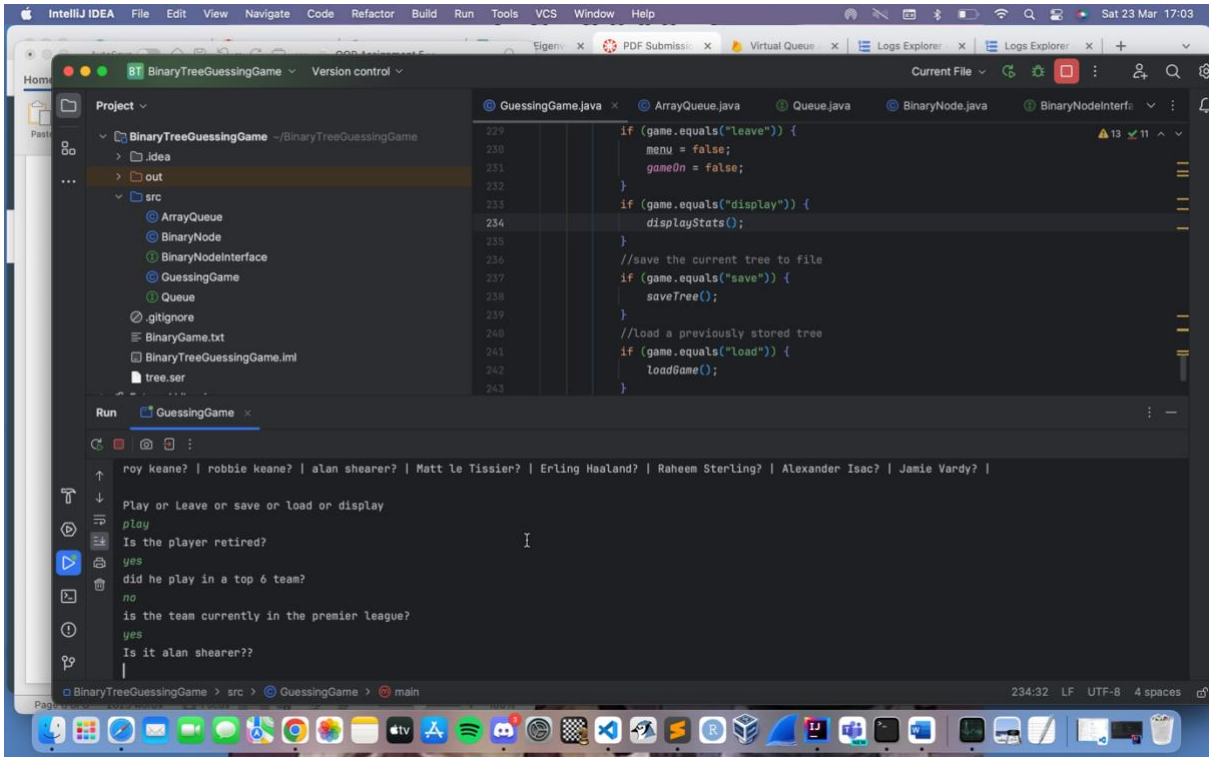
Testing.

Display stats of original tree:

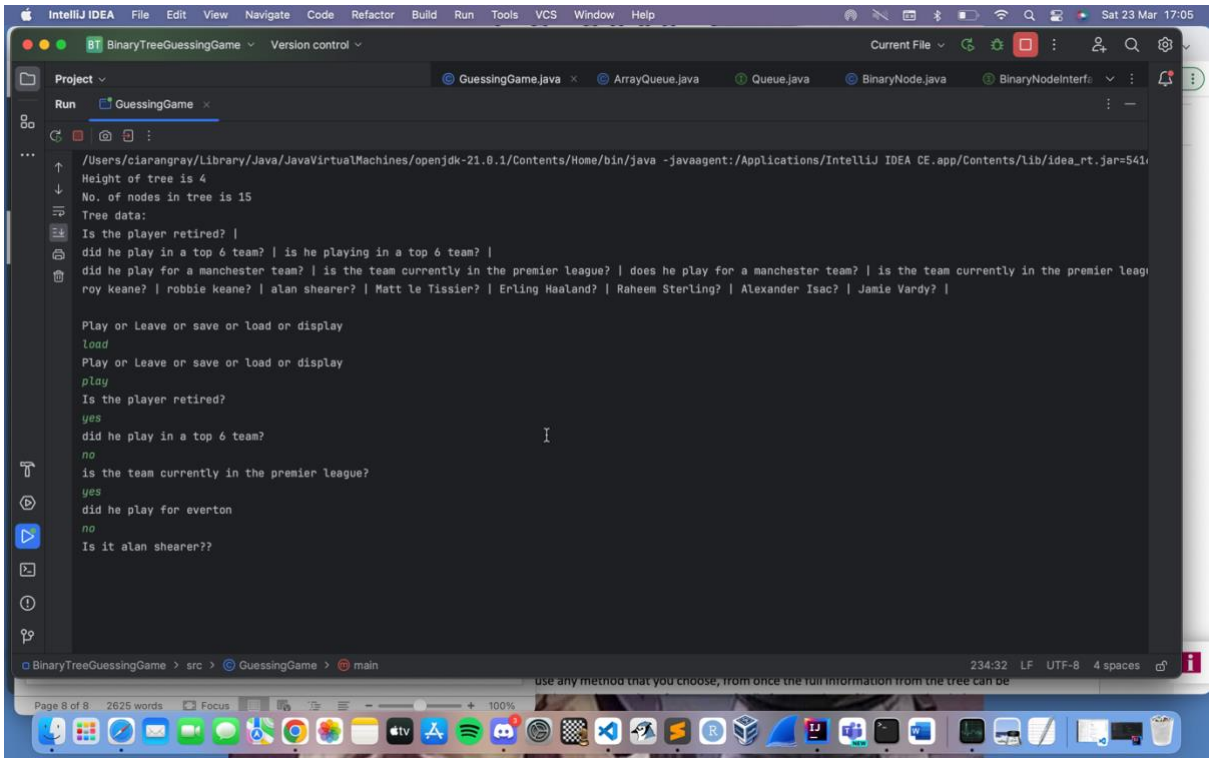


Difference between playing normal tree and playing a loaded tree:

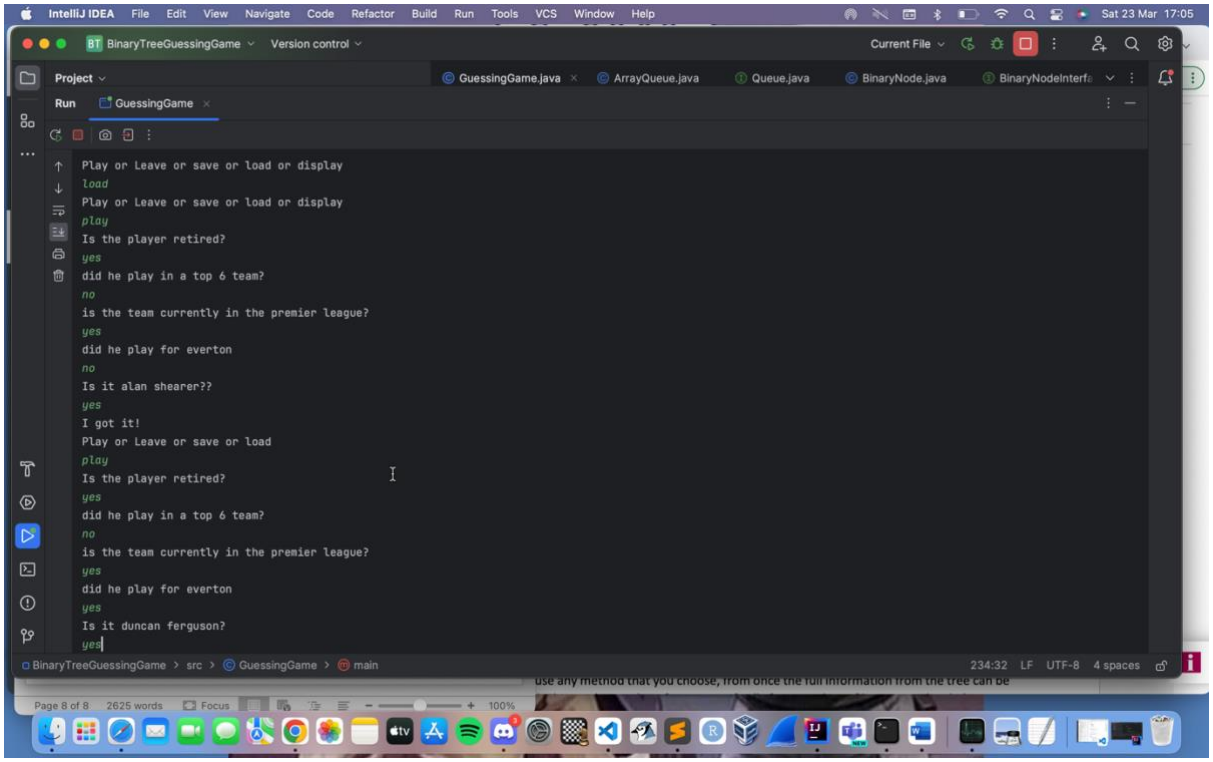
normal tree:



Loaded tree with same answer:

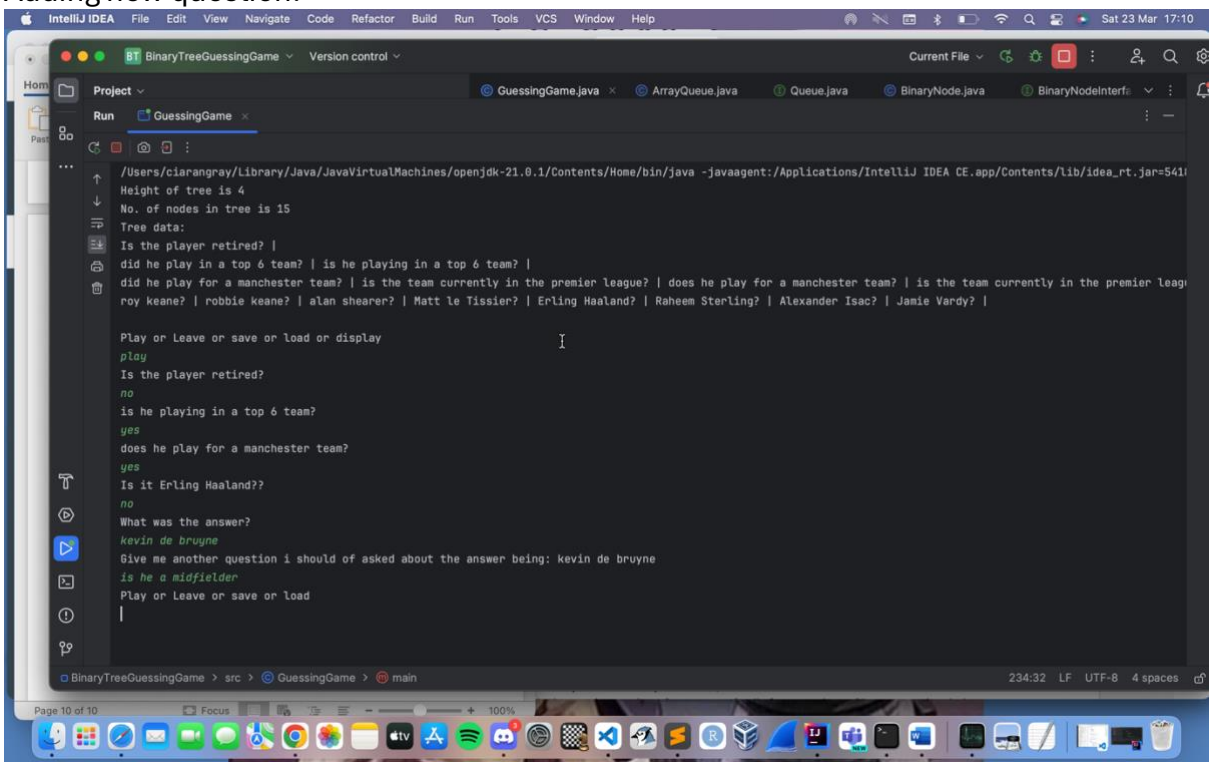


Loaded tree with new answer:

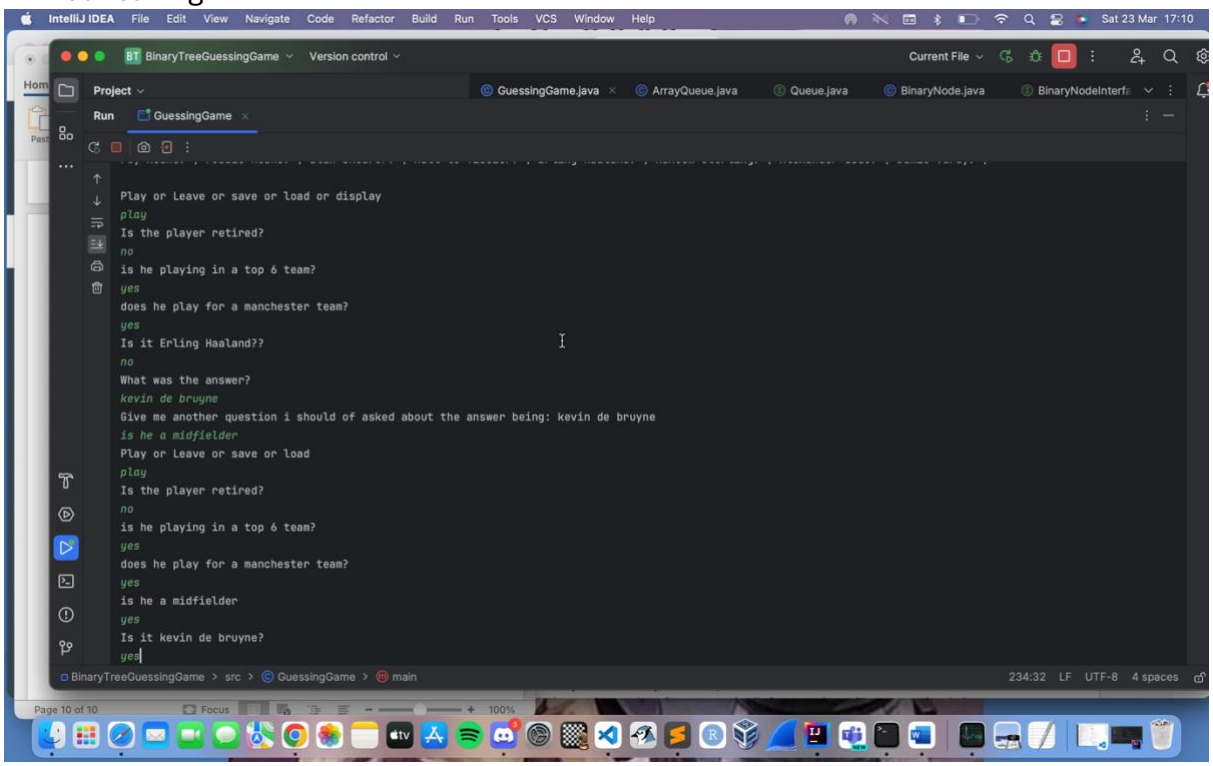


Adding new question onto a tree after incorrect answer:

Adding new question:

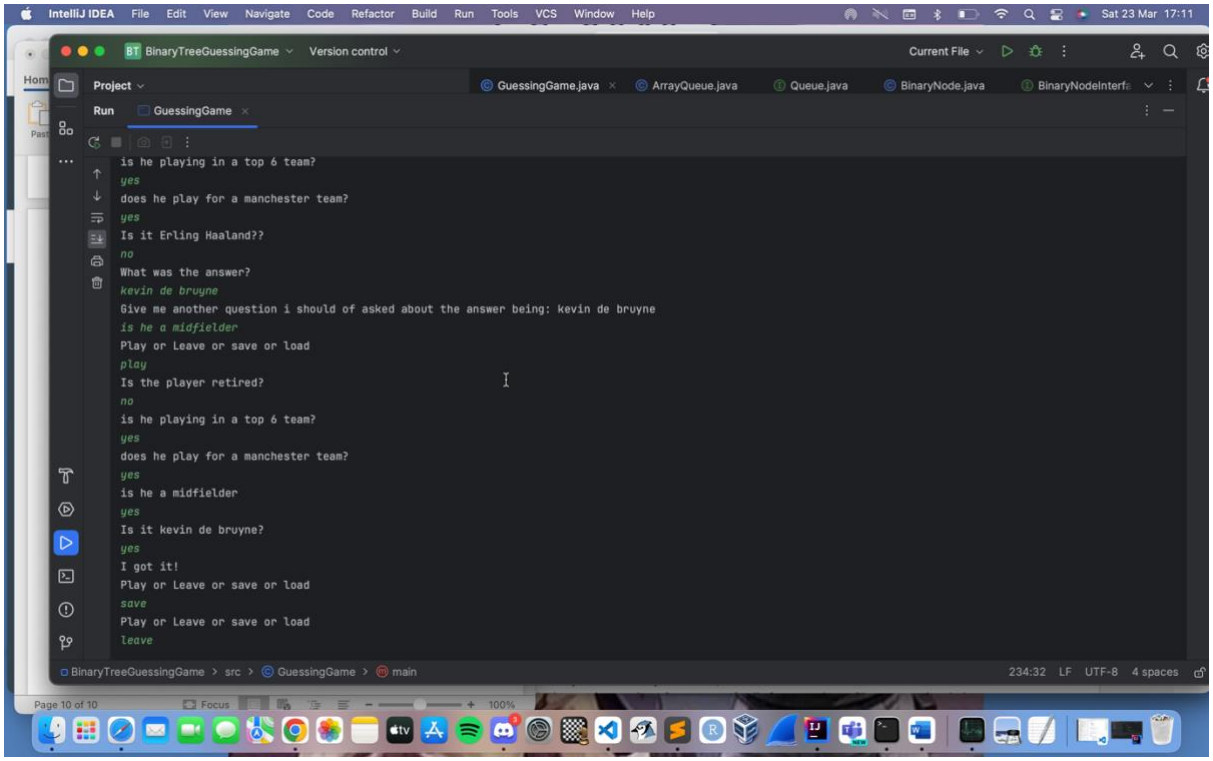


Getting the new question and answer to show in the tree in the same game session without saving:

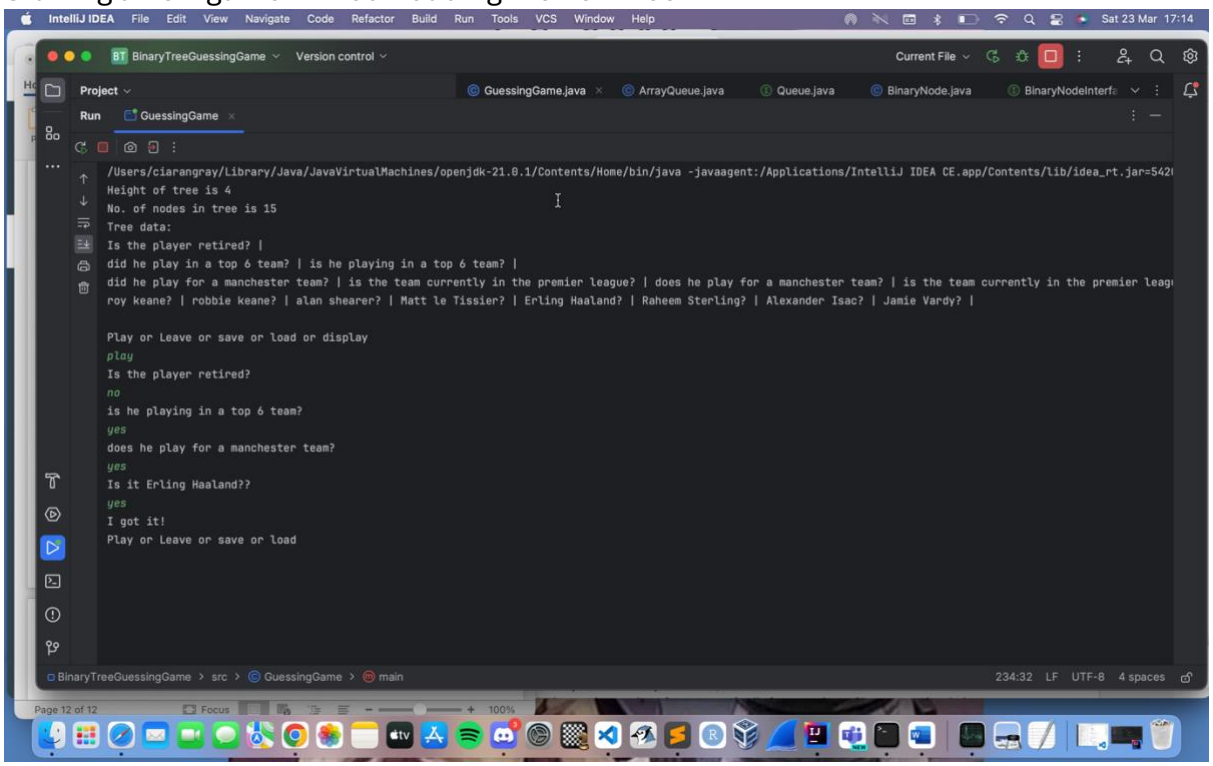


Saving a tree and loading it:

saving the tree with the new questions we added earlier:



Starting a new game without loading the new tree:



Loading the saved tree created above, choosing “no” this time to show old guess is still in the tree: