# CS 333 Project 1 Report

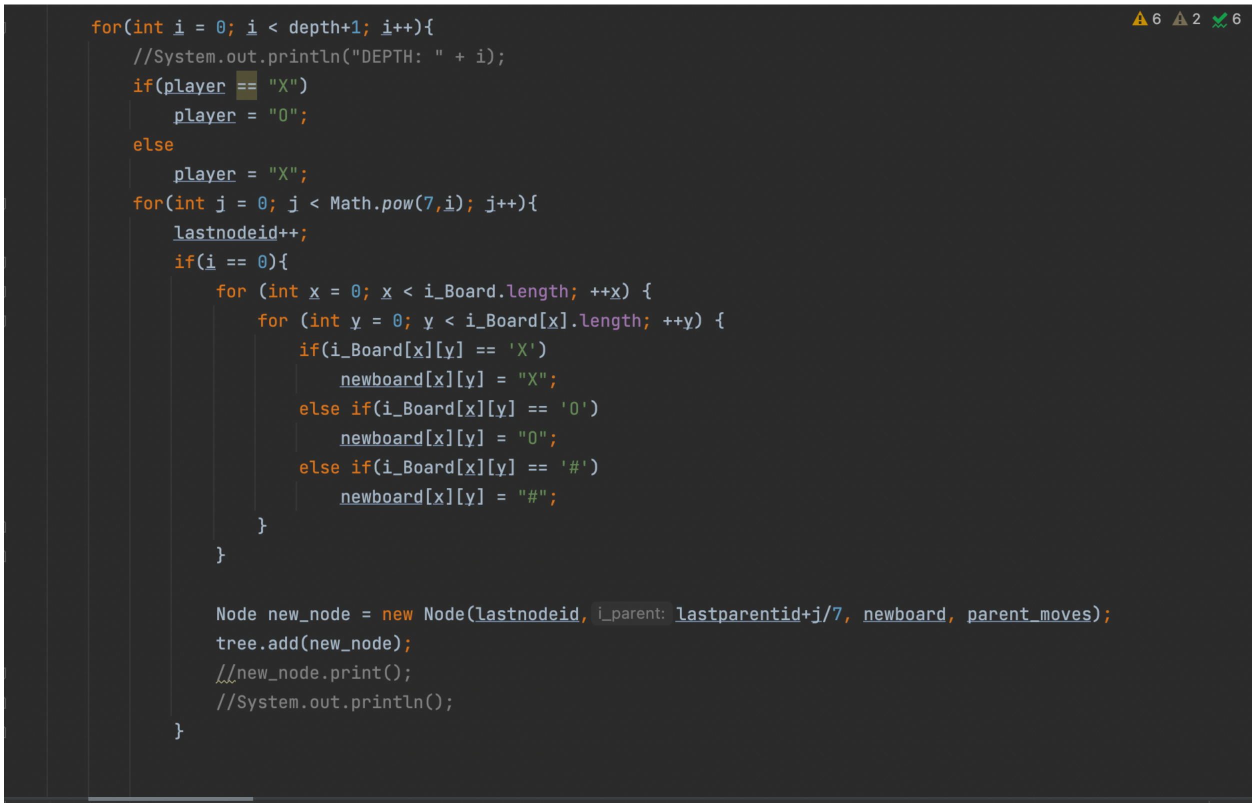
Utku Çelebiöven S017965

**Main Class (Connect4)**

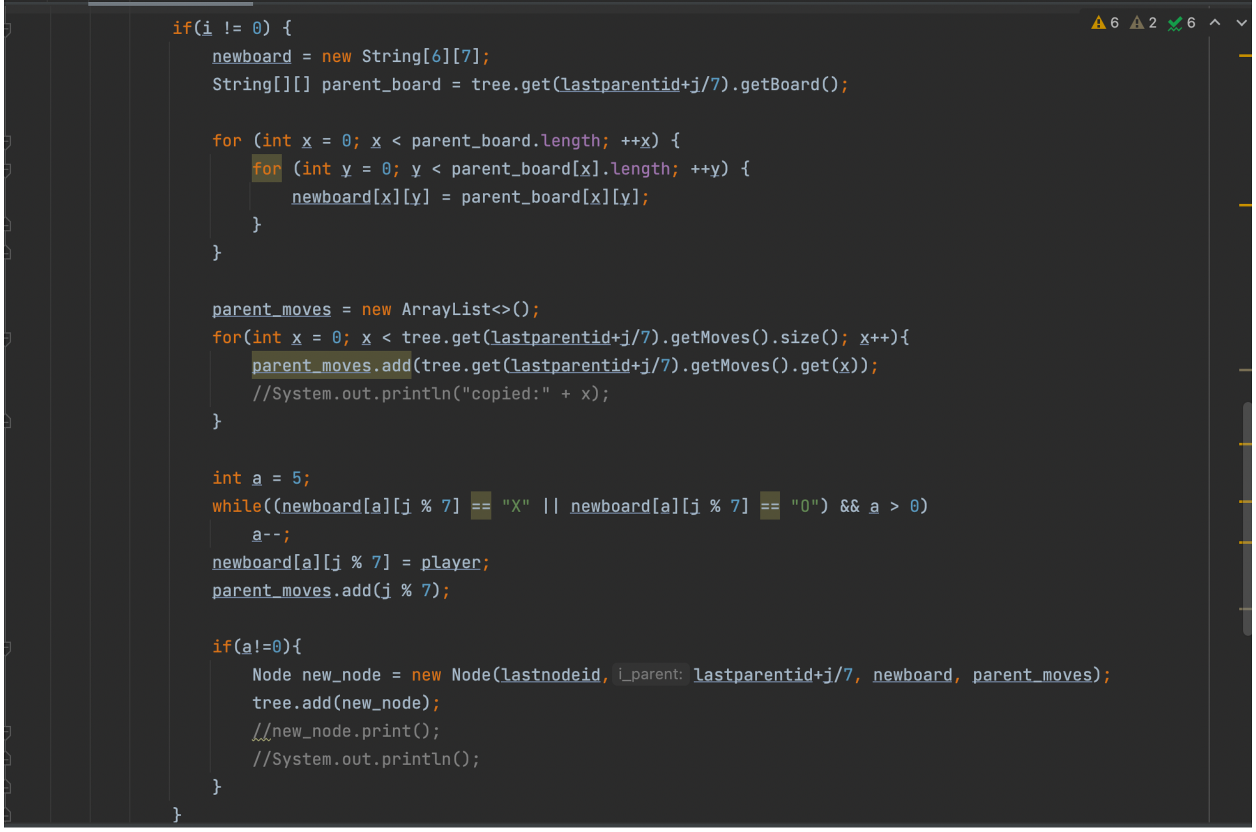
This class has a main function that runs the connect 4 game. The game board is created, edited, and checked inside this function. The game keeps going until one of the players win or a total of 42 moves have been made. For each play the computer needs to make, the main function calls the CreateTree class and pass the current state of the board. Then a chosen move is returned and played on the board. When its turn for the human player, the function waits for an input from the user. This function is also responsible for checking if there is a winner.

**Create Tree Class**

This class runs every time it is the computer's turn. It creates a tree of nodes that include all the possible plays available. The tree is created according to the depth input from the user. While creating this tree Node Class is used as a template to create nodes, which is explained in more detail in the next section. While creating a tree the nodes are stored in a list and connected to each other by parent information stored inside the node. After creating this tree, this class sends this tree into EvaluateTree function in order to find the best node to follow. Then it takes the move from the chosen node and send the move back to the game.



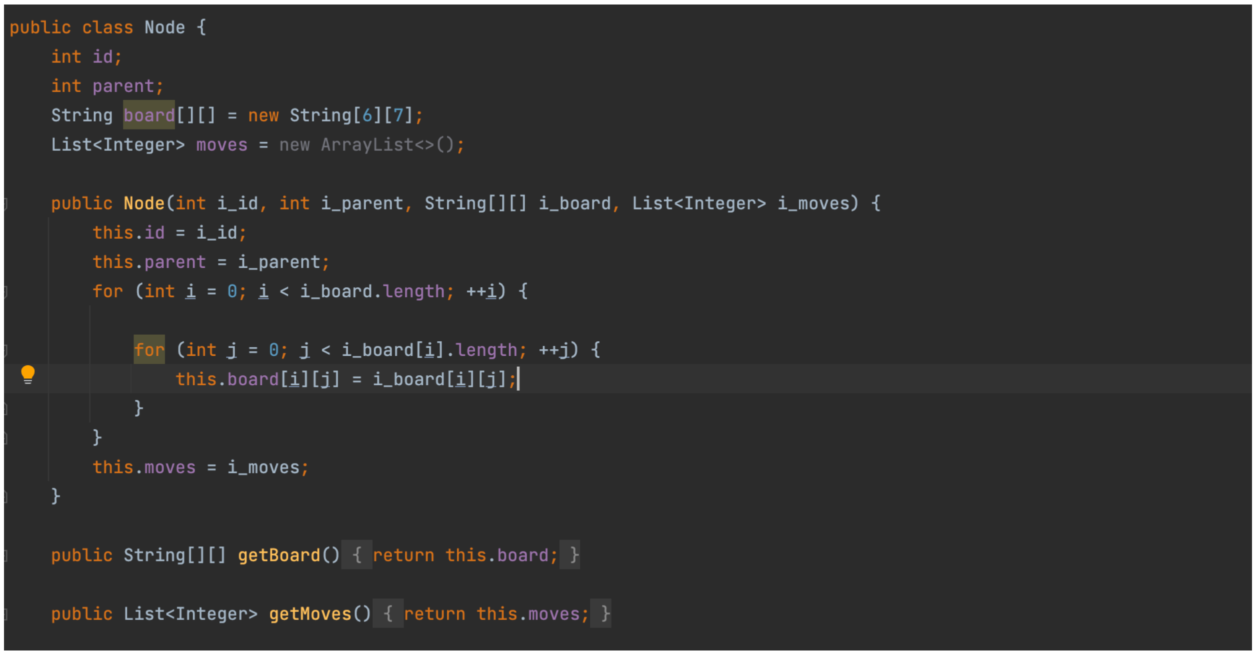
```java
for(int i = 0; i < depth+1; i++){
    //System.out.println("DEPTH: " + i);
    if(player == "X")
        player = "O";
    else
        player = "X";
    for(int j = 0; j < Math.pow(7,i); j++){
        lastnodeid++;
        if(i == 0){
            for (int x = 0; x < i_Board.length; ++x) {
                for (int y = 0; y < i_Board[x].length; ++y) {
                    if(i_Board[x][y] == 'X')
                        newboard[x][y] = "X";
                    else if(i_Board[x][y] == 'O')
                        newboard[x][y] = "O";
                    else if(i_Board[x][y] == '#')
                        newboard[x][y] = "#";
                }
            }

            Node new_node = new Node(lastnodeid, i_parent: lastparentid+j/7, newboard, parent_moves);
            tree.add(new_node);
            //new_node.print();
            //System.out.println();
        }
```



```java
        if(i != 0) {
            newboard = new String[6][7];
            String[][] parent_board = tree.get(lastparentid+j/7).getBoard();

            for (int x = 0; x < parent_board.length; ++x) {
                for (int y = 0; y < parent_board[x].length; ++y) {
                    newboard[x][y] = parent_board[x][y];
                }
            }

            parent_moves = new ArrayList<>();
            for(int x = 0; x < tree.get(lastparentid+j/7).getMoves().size(); x++){
                parent_moves.add(tree.get(lastparentid+j/7).getMoves().get(x));
                //System.out.println("copied:" + x);
            }

            int a = 5;
            while((newboard[a][j % 7] == "X" || newboard[a][j % 7] == "O") && a > 0)
                a--;
            newboard[a][j % 7] = player;
            parent_moves.add(j % 7);

            if(a!=0){
                Node new_node = new Node(lastnodeid, i_parent: lastparentid+j/7, newboard, parent_moves);
                tree.add(new_node);
                //new_node.print();
                //System.out.println();
            }
        }
```

## Node Class

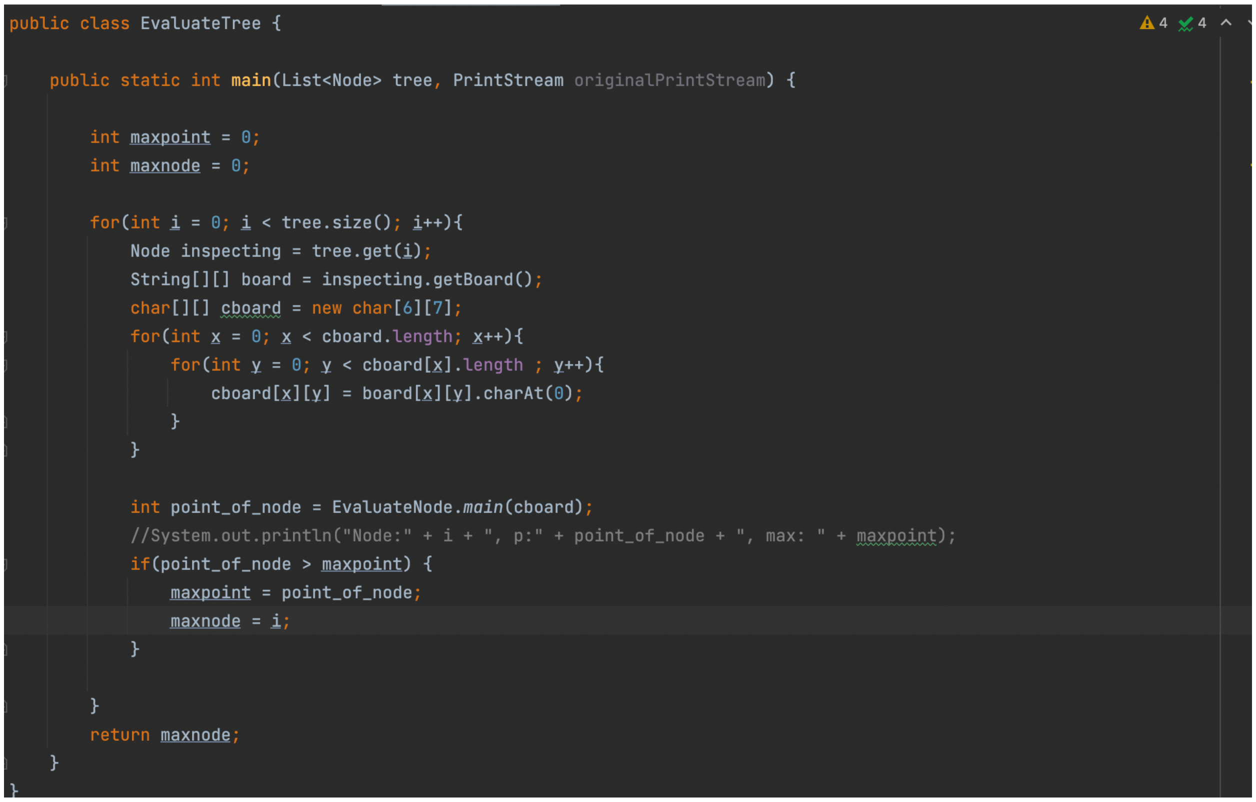
This class defines a template for nodes used inside the tree. Each node has id, parent, board and moves variables. Id variable defines the id of the node inside the tree while parent variable stores the id of the parent of that node. The board variable is a 2D string array which holds the board state in that node and the moves variable is a list of moves that holds which columns were played in order to get from the source node into this node.



```java
public class Node {
    int id;
    int parent;
    String board[][] = new String[6][7];
    List<Integer> moves = new ArrayList<>();

    public Node(int i_id, int i_parent, String[][] i_board, List<Integer> i_moves) {
        this.id = i_id;
        this.parent = i_parent;
        for (int i = 0; i < i_board.length; ++i) {

            for (int j = 0; j < i_board[i].length; ++j) {
                this.board[i][j] = i_board[i][j];
            }
        }
        this.moves = i_moves;
    }

    public String[][] getBoard() { return this.board; }

    public List<Integer> getMoves() { return this.moves; }
```

## Evaluate Tree Class

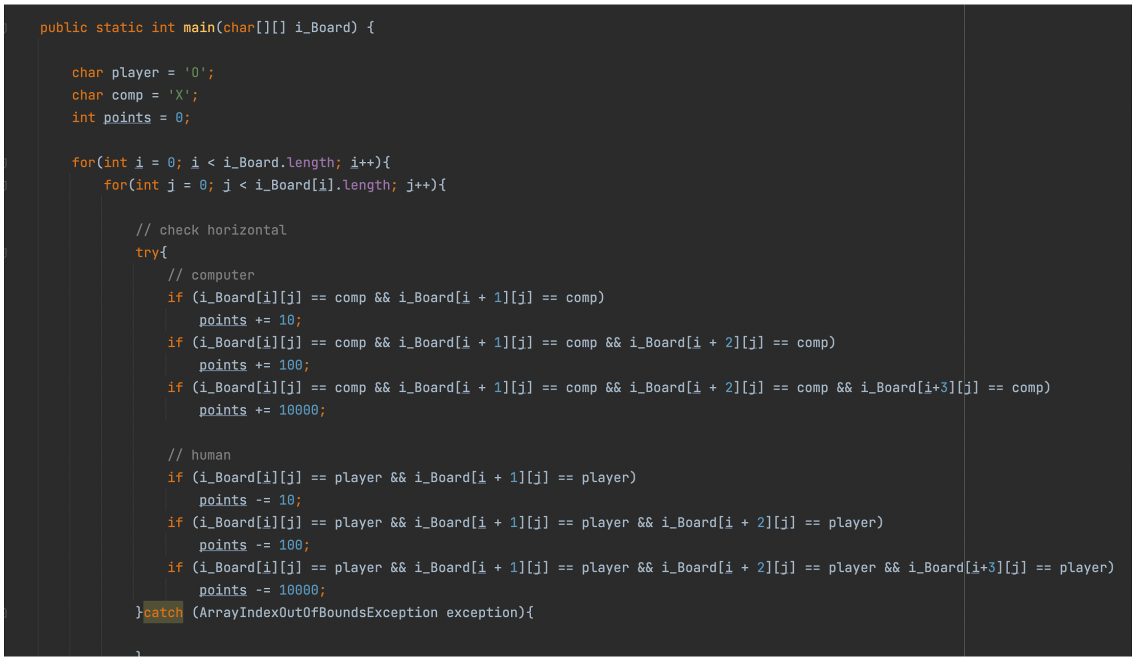
This class processes the tree created inside the CreateTree class and it performs a Breadth First Search on the tree. One by one it checks every node in each depth, meaning it checks the other children of a node before going a level deeper. While checking a node it sends the board of that node in the tree into the EvaluateNode class and gets back a score for that node. Then it keeps track of the node with the best score in "maxnode" variable and returns that node's id. I preferred BFS over DFS because I think evaluating alternative options before going deeper into one option is a better strategy for this game.



```java
public class EvaluateTree {

    public static int main(List<Node> tree, PrintStream originalPrintStream) {

        int maxpoint = 0;
        int maxnode = 0;

        for(int i = 0; i < tree.size(); i++){
            Node inspecting = tree.get(i);
            String[][] board = inspecting.getBoard();
            char[][] cboard = new char[6][7];
            for(int x = 0; x < cboard.length; x++){
                for(int y = 0; y < cboard[x].length ; y++){
                    cboard[x][y] = board[x][y].charAt(0);
                }
            }

            int point_of_node = EvaluateNode.main(cboard);
            //System.out.println("Node:" + i + ", p:" + point_of_node + ", max: " + maxpoint);
            if(point_of_node > maxpoint) {
                maxpoint = point_of_node;
                maxnode = i;
            }

        }
        return maxnode;
    }
}
```

## Evaluate Node Class

This class evaluates the given board in respect of the players and returns a score which indicates the success possibility of that node. It scans the board one by one and checks if there are 2, 3 or 4 pieces together. If there are and the pieces belong to computer it adds a positive score accordingly and if the pieces were the humans then it adds a negative score. Then the total score is returned.



```java
public static int main(char[][] i_Board) {

    char player = 'O';
    char comp = 'X';
    int points = 0;

    for(int i = 0; i < i_Board.length; i++){
        for(int j = 0; j < i_Board[i].length; j++){

            // check horizontal
            try{
                // computer
                if (i_Board[i][j] == comp && i_Board[i + 1][j] == comp)
                    points += 10;
                if (i_Board[i][j] == comp && i_Board[i + 1][j] == comp && i_Board[i + 2][j] == comp)
                    points += 100;
                if (i_Board[i][j] == comp && i_Board[i + 1][j] == comp && i_Board[i + 2][j] == comp && i_Board[i+3][j] == comp)
                    points += 10000;

                // human
                if (i_Board[i][j] == player && i_Board[i + 1][j] == player)
                    points -= 10;
                if (i_Board[i][j] == player && i_Board[i + 1][j] == player && i_Board[i + 2][j] == player)
                    points -= 100;
                if (i_Board[i][j] == player && i_Board[i + 1][j] == player && i_Board[i + 2][j] == player && i_Board[i+3][j] == player)
                    points -= 10000;
            }catch (ArrayIndexOutOfBoundsException exception){
```