Tic-Tac-Toe & Connect Four Game

FINAL PROJECT

Rahul Pothirendi, Rohit Varma Mudundi
Software Engineering Systems, Northeastern University

pothirendi.r@northeastern.edu, mudundi.r@northeastern.edu

Abstract – This project focuses on creating a Connect Four game using Monte Carlo Tree Search (MCTS). The report discusses how the idea for the game originated and the challenges encountered during development. It explains the importance of MCTS in the project and how Tic-Tac-Toe served as a model for building the Connect Four game.

PROBLEM DESCRIPTION

In this project, we're tasked with developing our own game using Monte Carlo Tree Search (MCTS), an algorithm for decision-making in certain types of games. MCTS involves four key steps:

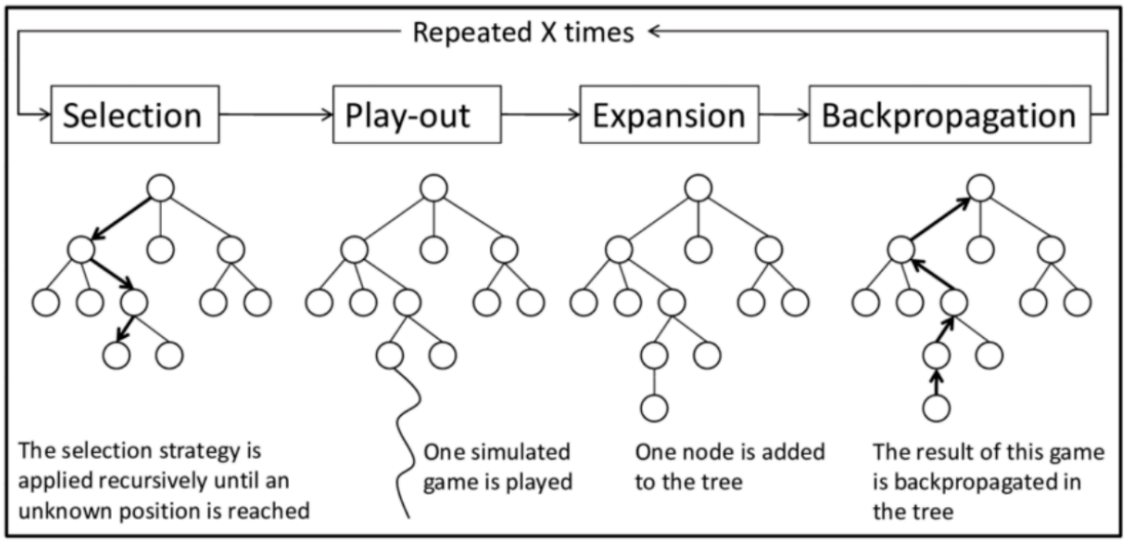
Selection, Expansion, Simulation, and Backpropagation.

Selection begins at the root node and navigates through the tree based on a selection policy until reaching an unexplored node.

Expansion involves adding child nodes to expand the tree.

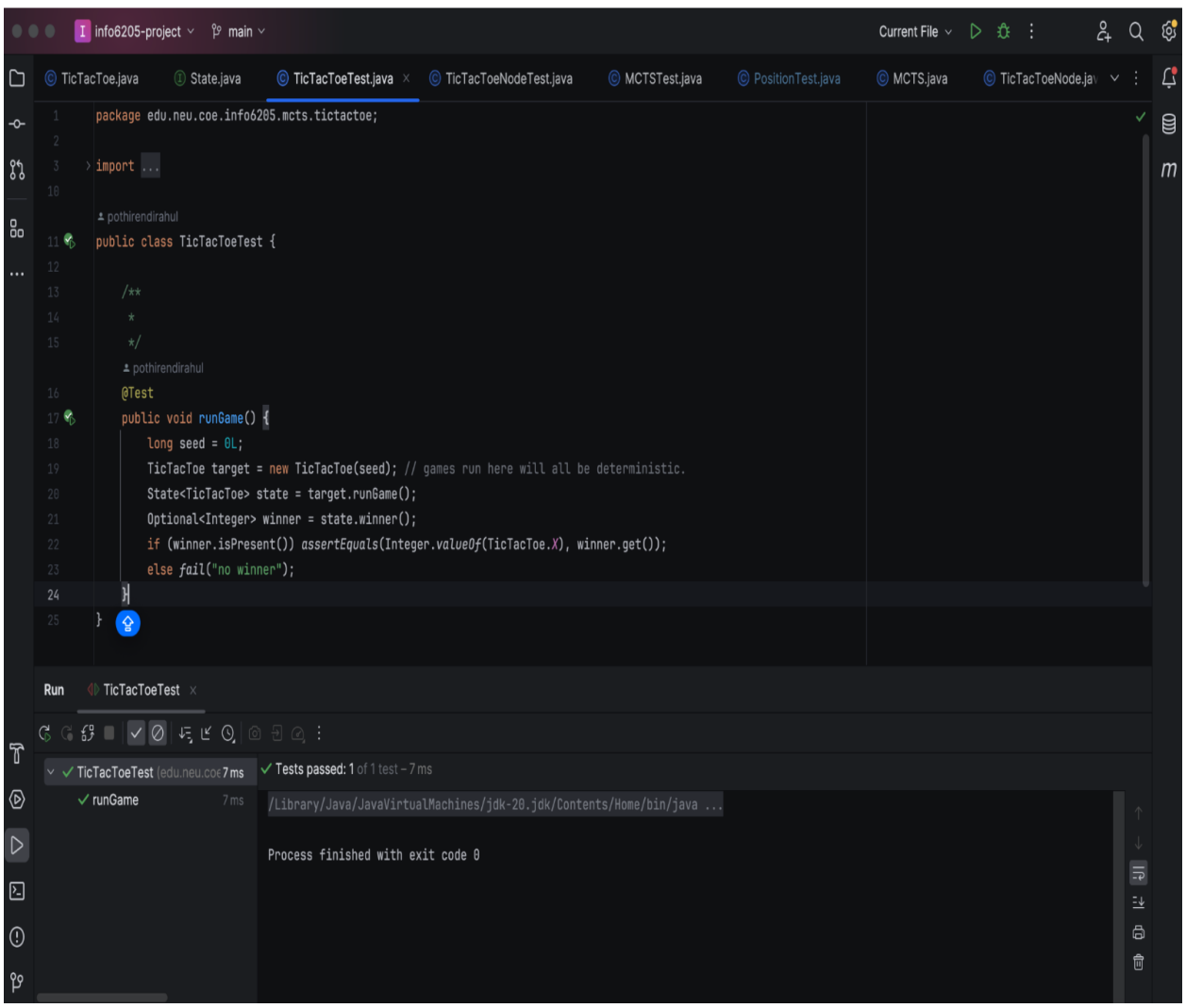
Simulation entails playing out the game from the current node to the end, recording results for all possible outcomes, and selecting the best solution.

Backpropagation involves updating the information stored in the nodes traversed to reach the terminal node. Understanding and implementing these four steps are essential for the search algorithm to function effectively.

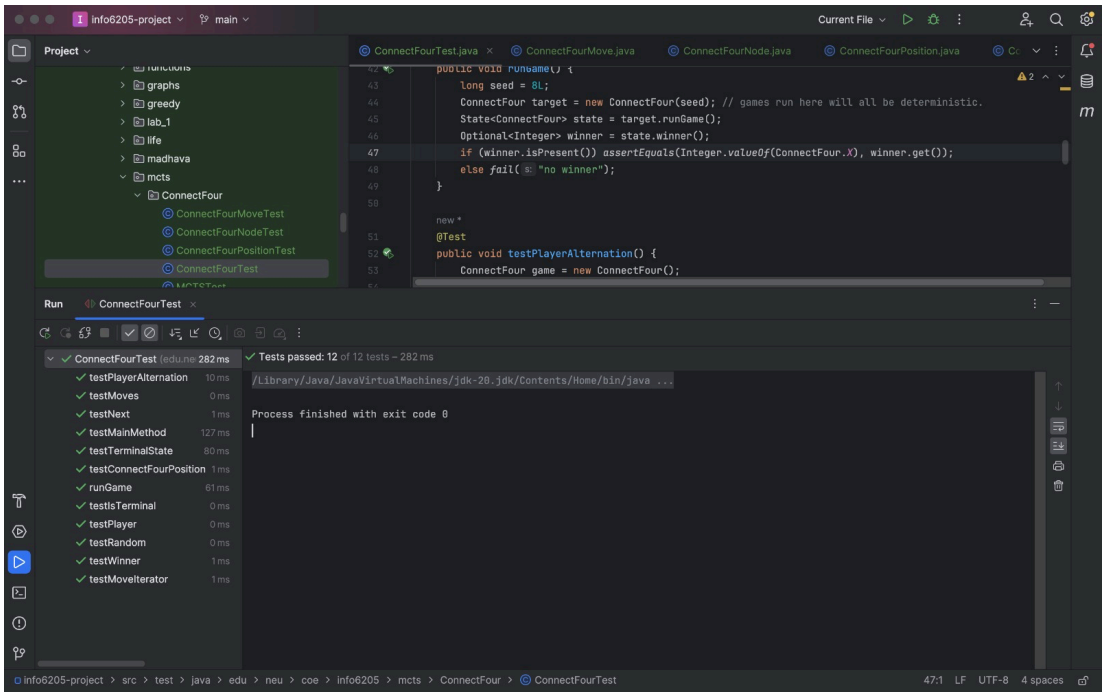


ANALYSIS

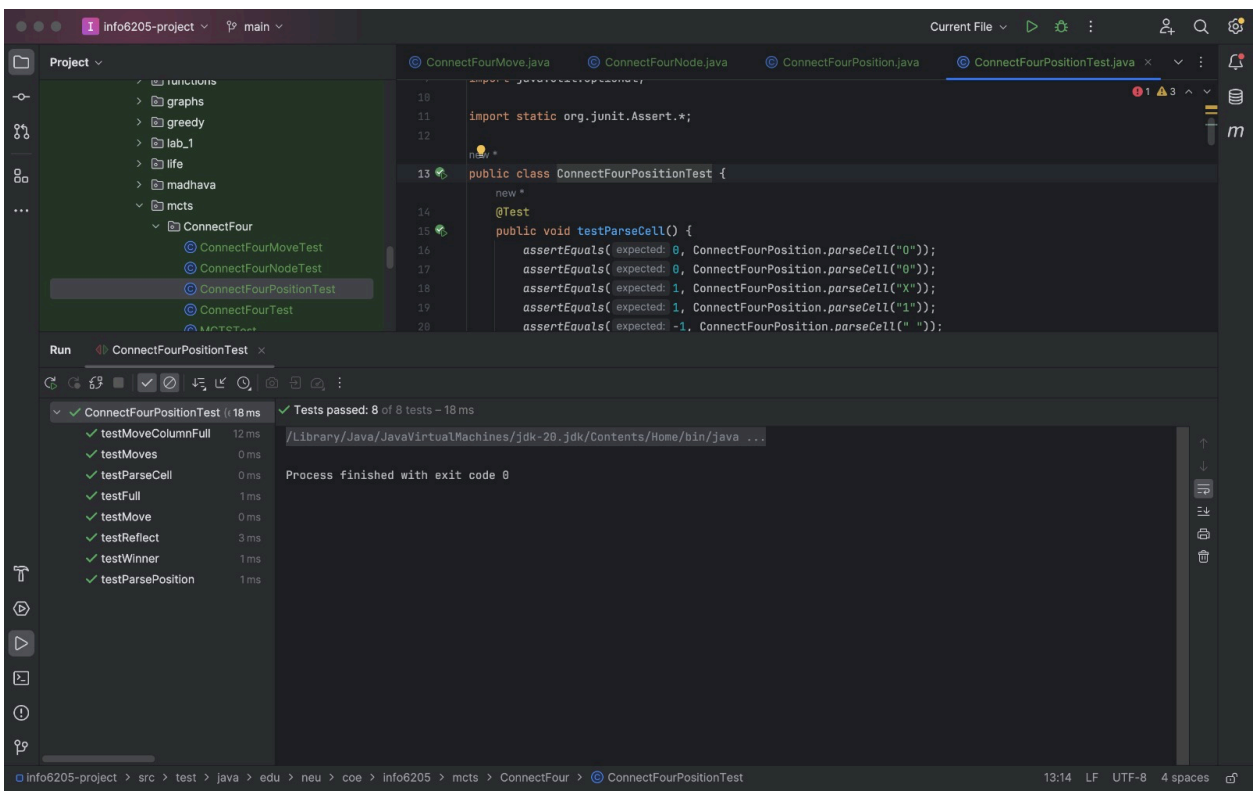
Before starting our main project, we practiced with Tic-Tac-Toe to understand how Monte Carlo Tree Search (MCTS) works in a game. In Tic-Tac-Toe, we focused on managing the game's state using a 3x3 grid. We defined empty spaces as -1, 'O' as 0, and 'X' as 1. We wrote code to generate possible moves and check for winning combinations in rows, columns, and diagonals.



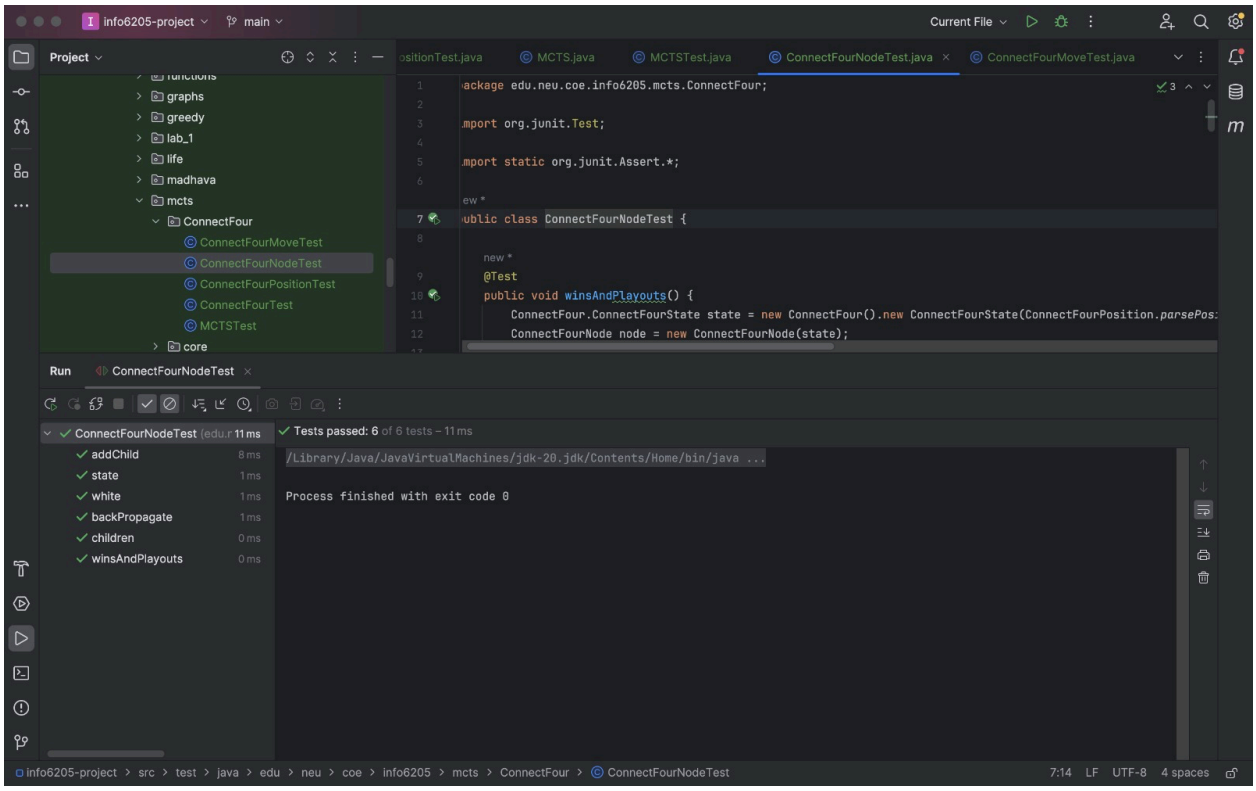
Unit test for TicTacToe



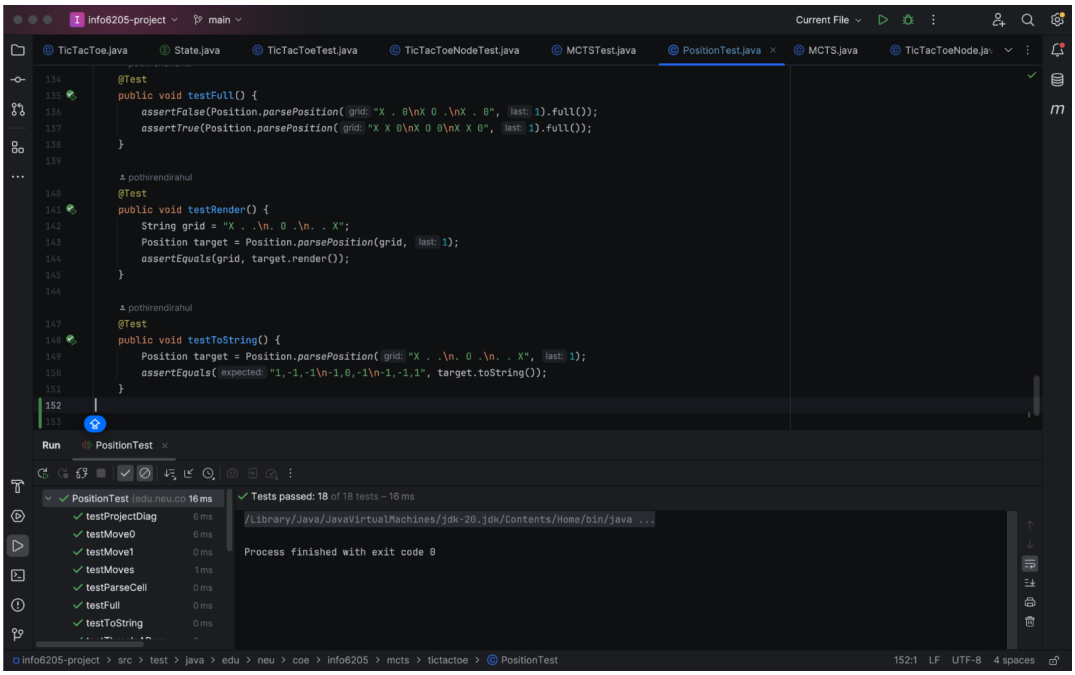
ConnectFour Test



ConnectFour Position Test

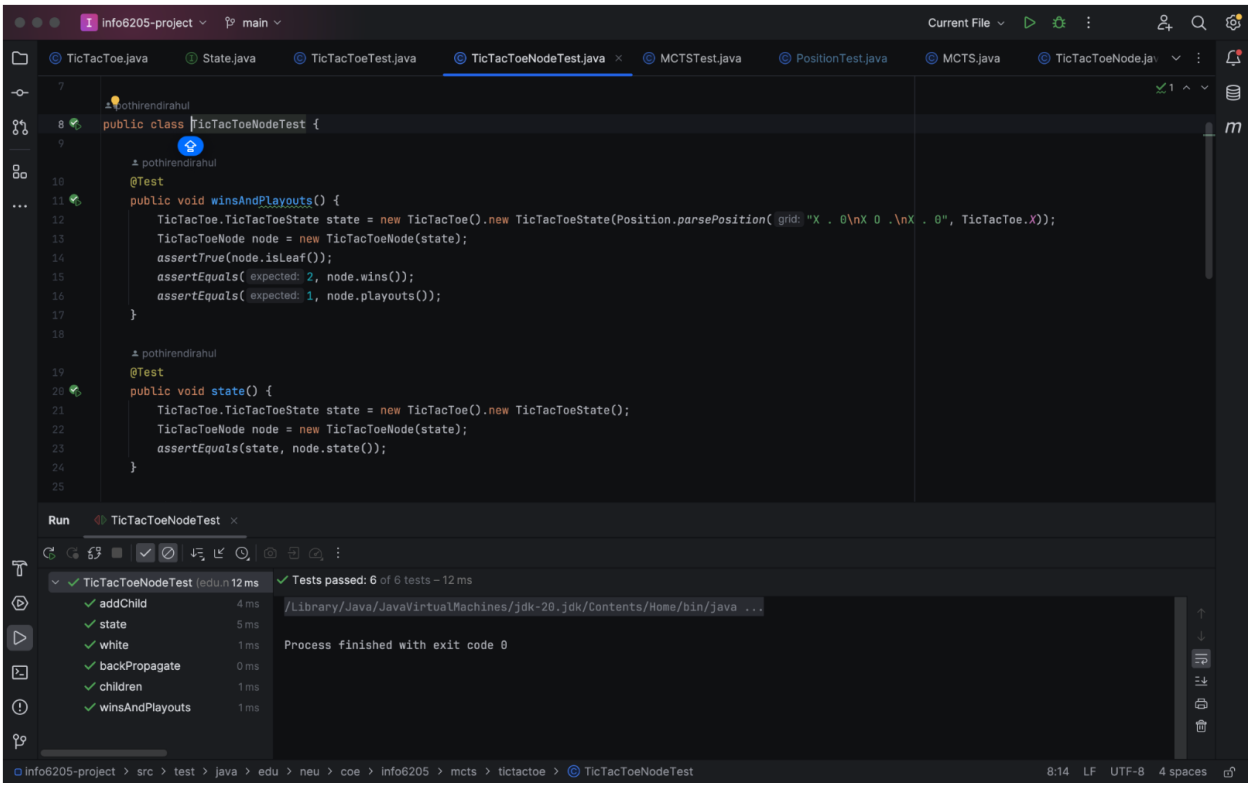


Connect Four Node Test



Unit test for PositionTest

Unit test for TicTacToc Node



IMPLEMENTATION

As we brainstormed our game concept, we considered developing a Connect Four game. The game would start with a grid where players can drop their tokens. The grid would have multiple columns and rows. Players take turns dropping their tokens into a column, and the tokens stack up vertically. The objective is to connect four of their tokens vertically, horizontally, or diagonally before their opponent does. For example, a player might choose column 3 and drop their token into it, causing the token to stack on top of any existing tokens in the column.

IMPLEMENTATION OF TICTACTOE.JAVA

```java
public static void main(String[] args) {
    int[] totalGames = new int[]{200, 400, 800, 1600, 3200, 6400};

    for (int n : totalGames) {
        runGames(n);
    }
}

public static void runGames(int totalGames) {
    int playerXCount = 0;
```

```
/Library/Java/JavaVirtualMachines/jdk-20.jdk/Contents/Home/bin/java ...
Total Games Player X (X) Wins   Player O (O) Wins   Draws
200     117         83          0
--------------------
Total Games Player X (X) Wins   Player O (O) Wins   Draws
400     329         52          19
--------------------
Total Games Player X (X) Wins   Player O (O) Wins   Draws
800     526         274         0
--------------------
Total Games Player X (X) Wins   Player O (O) Wins   Draws
1600        826         461         313
--------------------
Total Games Player X (X) Wins   Player O (O) Wins   Draws
3200        1799            903         498
--------------------
Total Games Player X (X) Wins   Player O (O) Wins   Draws
6400        3946            1326            1128
--------------------

Process finished with exit code 0
```

```java
package edu.neu.coe.info6205.mcts.tictactoe;

import edu.neu.coe.info6205.mcts.core.Node;

/**
 * Class to represent a Monte Carlo Tree Search for TicTacToe.
 */
public class MCTS {

    public static void main(String[] args) {
        MCTS mcts = new MCTS(new TicTacToeNode(new TicTacToe().new TicTacToeState()));
        Node<TicTacToe> root = mcts.root;


        int iterations = 1200;
        mcts.runMCTS(iterations, root);
    }

    public MCTS(Node<TicTacToe> root) {
        this.root = root;
    }

    private final Node<TicTacToe> root;


    public void runMCTS(int iterations, Node<TicTacToe> root) {
        for (int i = 0; i < iterations; i++) {
            TicTacToeNode promisingNode = selectPromisingNode((TicTacToeNode) root);
```

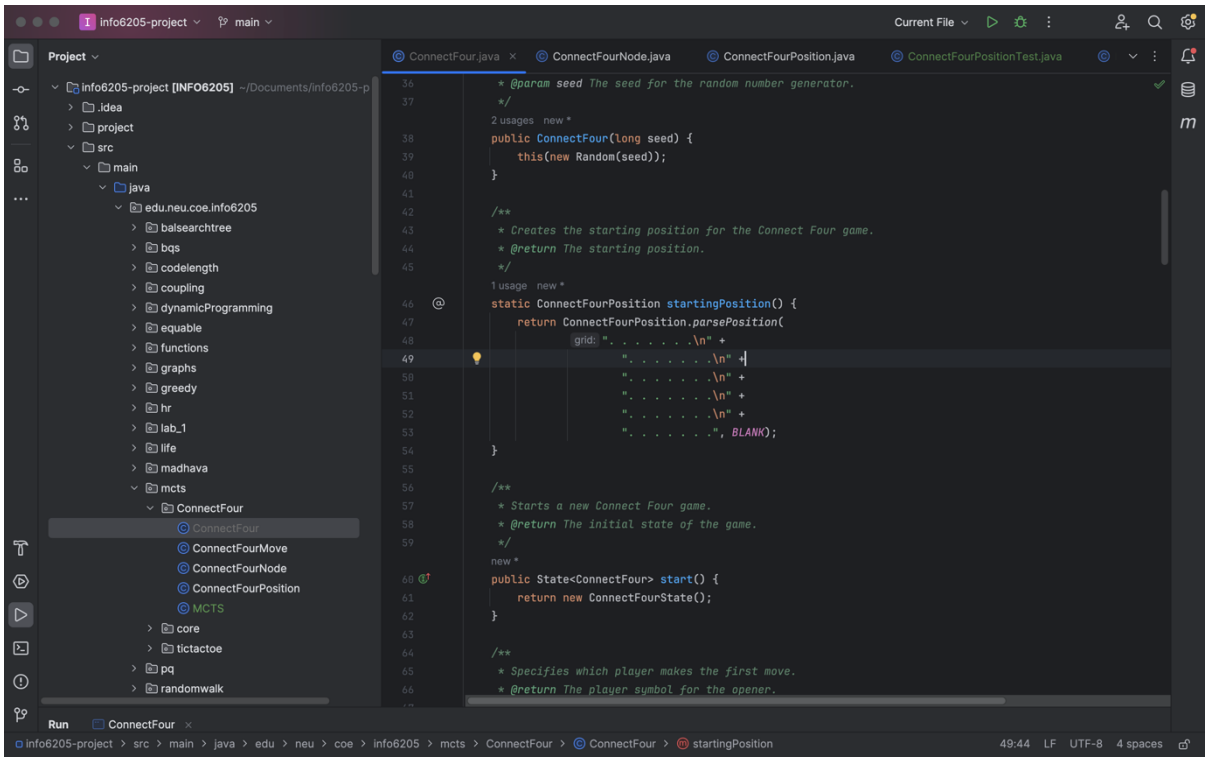
MCTS TicTacToe

```
. X . . X . O
. X . O X . X
. O . X X O O
O O . X O O X
X X O O X X O
O O O X X X O

. X . . X . O
. X . O X . X
. O . X X O O
O O . X O O X
X X O O X X O
O O O X X X O

ConnectFour: Winner is: O (Random)
ConnectFour: The Winner is: O

Process finished with exit code 0
```



EVALUATION

With these implementations in place, we tested our game to ensure its functionality. The screenshot below depicts the appearance of the Connect Four game, with the grid correctly generated and the game logic functioning as intended. Additionally, other sections of the program include unit tests to verify the proper execution of methods.

Total Games	Player X (X) Wins	Player O (O) Wins	Draws
200	104	78	18
400	234	166	0
800	381	387	32
1600	703	660	237
3200	1850	988	362
6400	3152	2122	1126

CONCLUSIONS

In this project, we recognize MCTS as a search algorithm utilized in decision-making games, enabling complex decision-making through Selection, Expansion, Simulation, and Backpropagation. Initially, we grasped the concept of MCTS by engaging with a sample Tic-Tac-Toe game. Subsequently, we delved further into MCTS by crafting our own Connect Four game.

REFERENCES

FOR TICTACTOE

https://www.baeldung.com/java-monte-carlo-tree-search

https://martin-ueding.de/posts/tic-tac-toe-with-monte-carlo-tree-search/

FOR MCTS CONNECT FOUR
https://ai.stackexchange.com/questions/21019/should-monte-carlo-tree-search-be-able-to-consistently-beat-me-in-the-connect-fo

https://link.springer.com/article/10.1007/s10462-022-10228-y