Program Structures and Algorithms Spring 2024

Team Member:

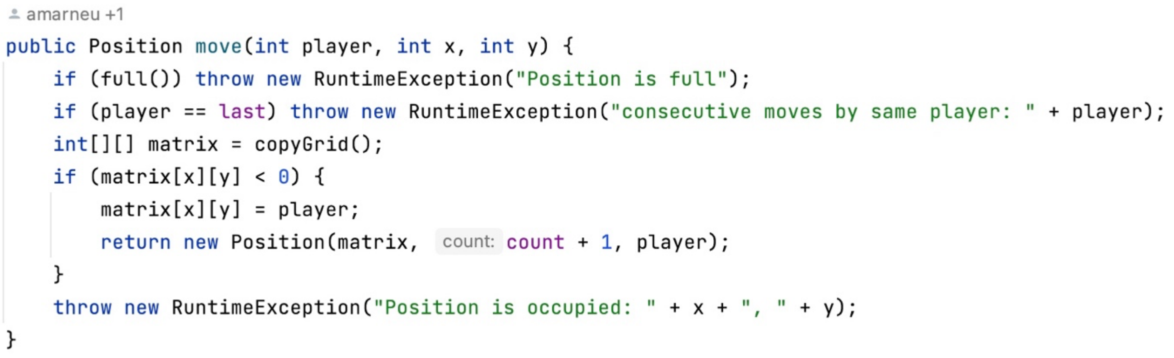
PRANAV ARUN KAPSE | NUID: 002871241
AMAR NAGARGOJE | NUID: 002273113
KHUSHANK MISTRY | NUID: 002209157

GitHub Link: https://github.com/amarneu/INFO6205-MCTS-FinalProject

TASK: TicTacToe Milestone 1

Aim: This milestone's main goal is to increase comprehension of the TicTacToe game's implementation through practical coding experience. The objective is to implement the code segments those requires in TicTacToe, to make sure the game runs properly. This entails writing code for gameplay elements like player movements, move possibilities, and winning scenarios. By doing this, students should be able to understand the general coding methodology that is applied in the architecture of the game, especially in regard to how the game logic is isolated from the details of game elements such as Game, Move, Node, or State.

Code Snippet:

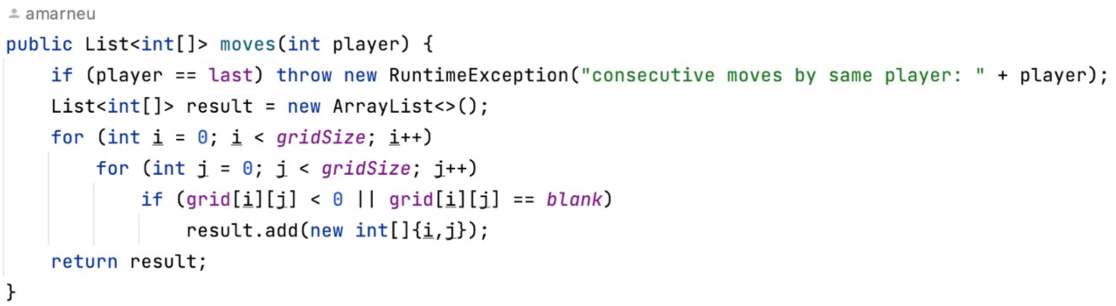


```
amarneu +1
public Position move(int player, int x, int y) {
    if (full()) throw new RuntimeException("Position is full");
    if (player == last) throw new RuntimeException("consecutive moves by same player: " + player);
    int[][] matrix = copyGrid();
    if (matrix[x][y] < 0) {
        matrix[x][y] = player;
        return new Position(matrix,  count: count + 1, player);
    }
    throw new RuntimeException("Position is occupied: " + x + ", " + y);
}
```

This method checks if the position is already occupied or not. If it's not occupied, it updates the grid with the player's move and returns a new Position object reflecting the updated state.

Moves:

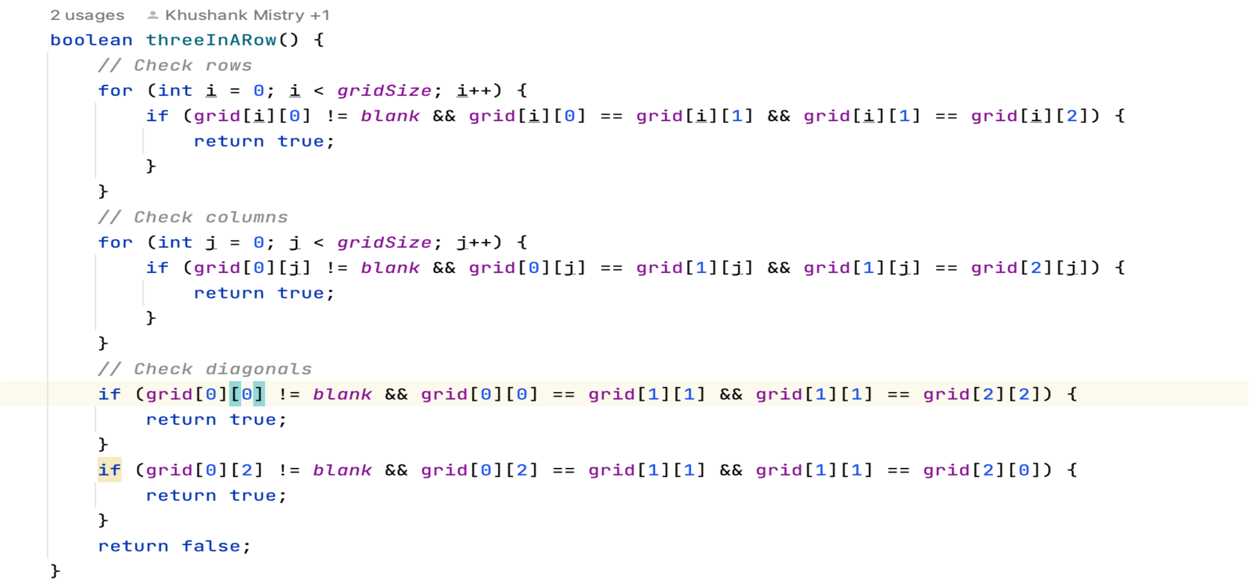
For the moves method in the Position class to yield all possible moves available for a player, you need to iterate over the grid and find empty cells. Here's how you can implement it:



amarneu

```java
public List<int[]> moves(int player) {
    if (player == last) throw new RuntimeException("consecutive moves by same player: " + player);
    List<int[]> result = new ArrayList<>();
    for (int i = 0; i < gridSize; i++)
        for (int j = 0; j < gridSize; j++)
            if (grid[i][j] < 0 || grid[i][j] == blank)
                result.add(new int[]{i,j});
    return result;
}
```

The threeInARow method in the Position class is responsible for determining if there are three consecutive cells with the same player symbol in a row, column, or diagonal. Here's how you can implement it:

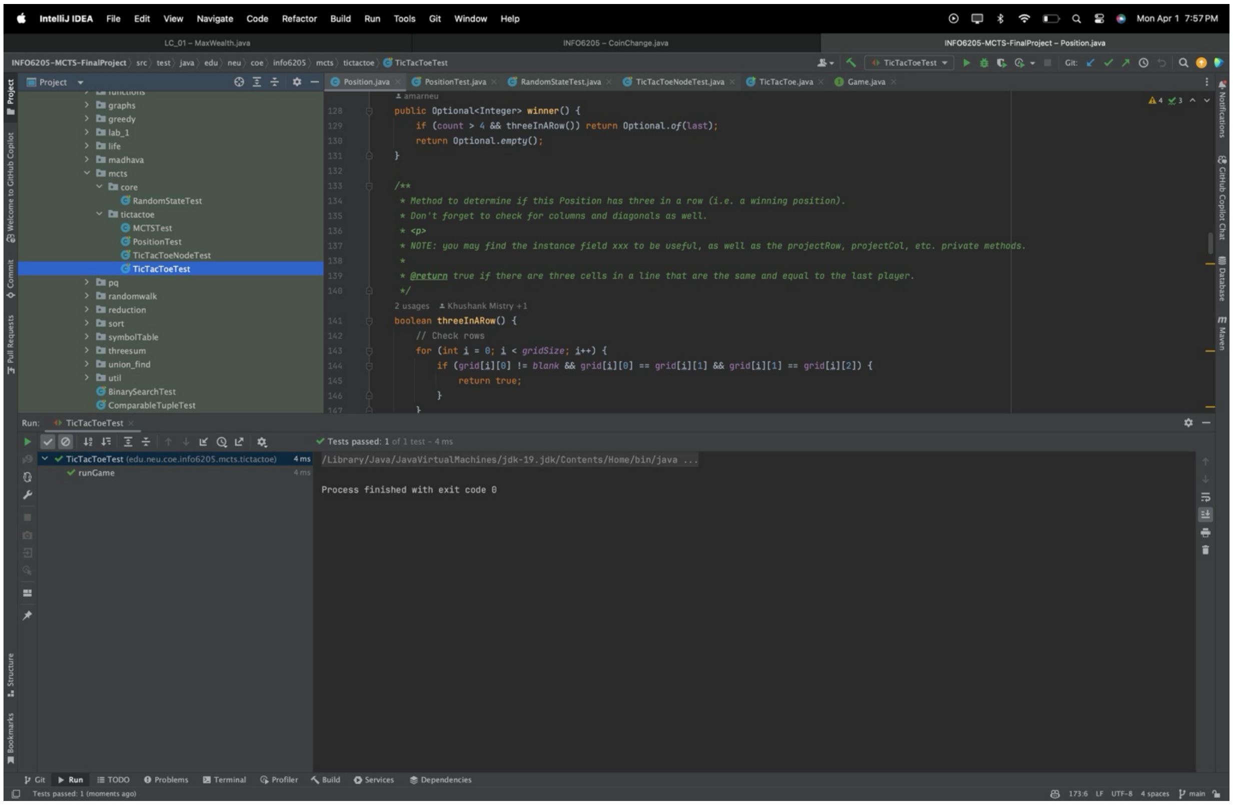


2 usages   Khushank Mistry +1

```java
boolean threeInARow() {
    // Check rows
    for (int i = 0; i < gridSize; i++) {
        if (grid[i][0] != blank && grid[i][0] == grid[i][1] && grid[i][1] == grid[i][2]) {
            return true;
        }
    }
    // Check columns
    for (int j = 0; j < gridSize; j++) {
        if (grid[0][j] != blank && grid[0][j] == grid[1][j] && grid[1][j] == grid[2][j]) {
            return true;
        }
    }
    // Check diagonals
    if (grid[0][0] != blank && grid[0][0] == grid[1][1] && grid[1][1] == grid[2][2]) {
        return true;
    }
    if (grid[0][2] != blank && grid[0][2] == grid[1][1] && grid[1][1] == grid[2][0]) {
        return true;
    }
    return false;
}
```

This method checks for three in a row in three different directions: rows, columns, and diagonals. If it finds three consecutive cells with the same player symbol in any of these directions, it returns true, indicating that there is a winner. Otherwise, it returns false.

Test Cases:

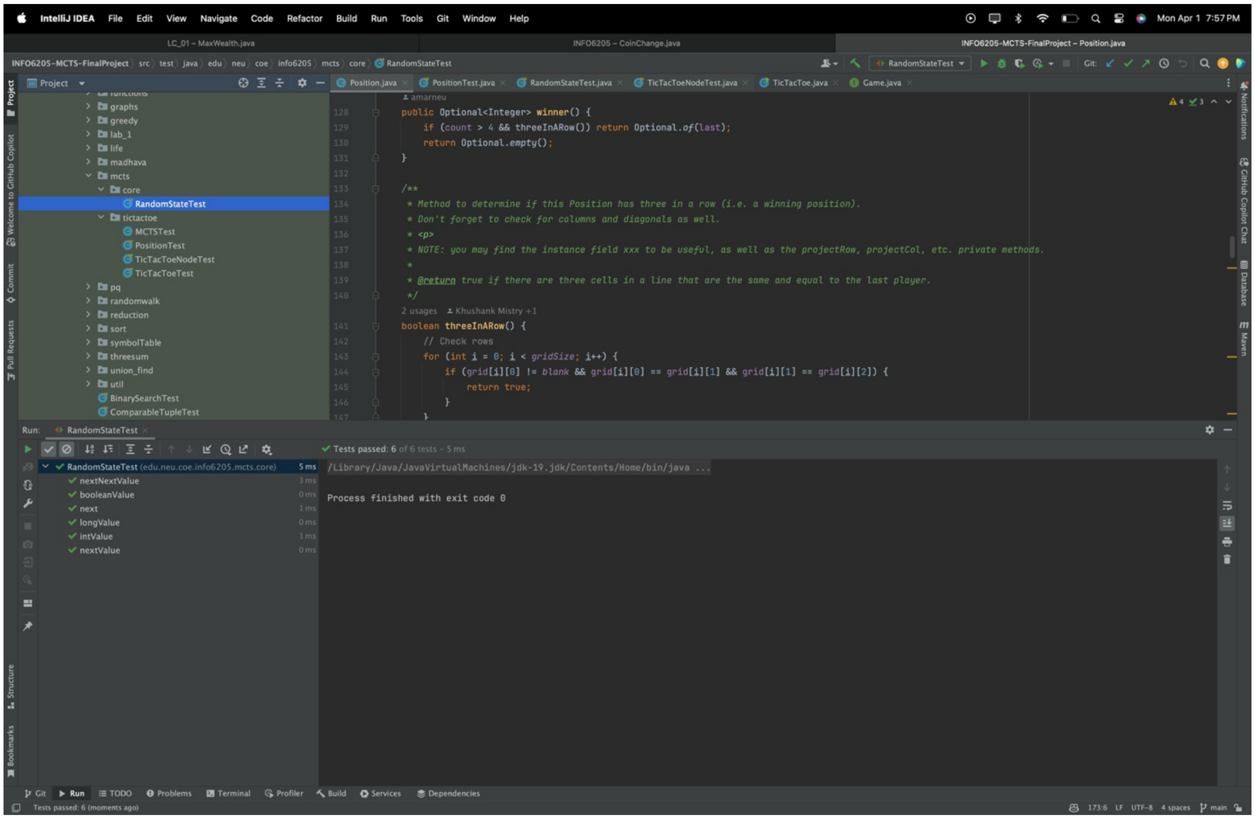


```java
public Optional<Integer> winner() {
    if (count > 4 && threeInARow()) return Optional.of(last);
    return Optional.empty();
}

/**
 * Method to determine if this Position has three in a row (i.e. a winning position).
 * Don't forget to check for columns and diagonals as well.
 * <p>
 * NOTE: you may find the instance field xxx to be useful, as well as the projectRow, projectCol, etc. private methods.
 *
 * @return true if there are three cells in a line that are the same and equal to the last player.
 */
boolean threeInARow() {
    // Check rows
    for (int i = 0; i < gridSize; i++) {
        if (grid[i][0] != blank && grid[i][0] == grid[i][1] && grid[i][1] == grid[i][2]) {
            return true;
        }
    }
```

Tests passed: 1 of 1 test – 4 ms

TicTacToeTest (edu.neu.coe.info6205.mcts.tictactoe) 4 ms
runGame 4 ms

```
/Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```



```
    amarneu
    public Optional<Integer> winner() {
        if (count > 4 && threeInARow()) return Optional.of(last);
        return Optional.empty();
    }

    /**
     * Method to determine if this Position has three in a row (i.e. a winning position).
     * Don't forget to check for columns and diagonals as well.
     * <p>
     * NOTE: you may find the instance field xxx to be useful, as well as the projectRow, projectCol, etc. private methods.
     *
     * @return true if there are three cells in a line that are the same and equal to the last player.
     */
    2 usages   Khushank Mistry +1
    boolean threeInARow() {
        // Check rows
        for (int i = 0; i < gridSize; i++) {
            if (grid[i][0] != blank && grid[i][0] == grid[i][1] && grid[i][1] == grid[i][2]) {
                return true;
            }
        }
```

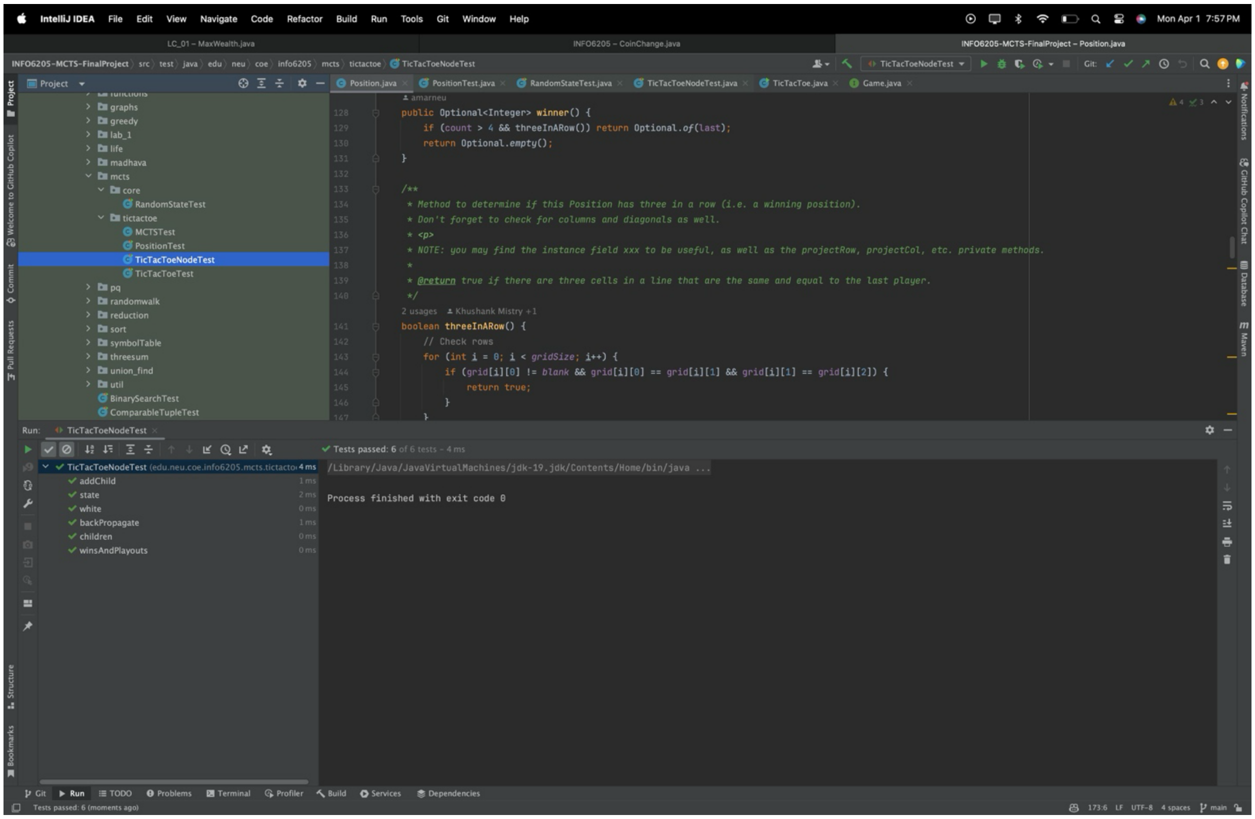
Run: RandomStateTest

✓ Tests passed: 6 of 6 tests – 5 ms

```
✓ RandomStateTest (edu.neu.coe.info6205.mcts.core)   5 ms
    ✓ nextNextValue   3 ms
    ✓ booleanValue    0 ms
    ✓ next            1 ms
    ✓ longValue       0 ms
    ✓ intValue        1 ms
    ✓ nextValue       0 ms
```

```
/Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

Tests passed: 6 (moments ago)



```
    amarneu
    public Optional<Integer> winner() {
        if (count > 4 && threeInARow()) return Optional.of(last);
        return Optional.empty();
    }

    /**
     * Method to determine if this Position has three in a row (i.e. a winning position).
     * Don't forget to check for columns and diagonals as well.
     * <p>
     * NOTE: you may find the instance field xxx to be useful, as well as the projectRow, projectCol, etc. private methods.
     *
     * @return true if there are three cells in a line that are the same and equal to the last player.
     */
    2 usages   Khushank Mistry +1
    boolean threeInARow() {
        // Check rows
        for (int i = 0; i < gridSize; i++) {
            if (grid[i][0] != blank && grid[i][0] == grid[i][1] && grid[i][1] == grid[i][2]) {
                return true;
            }
        }
```

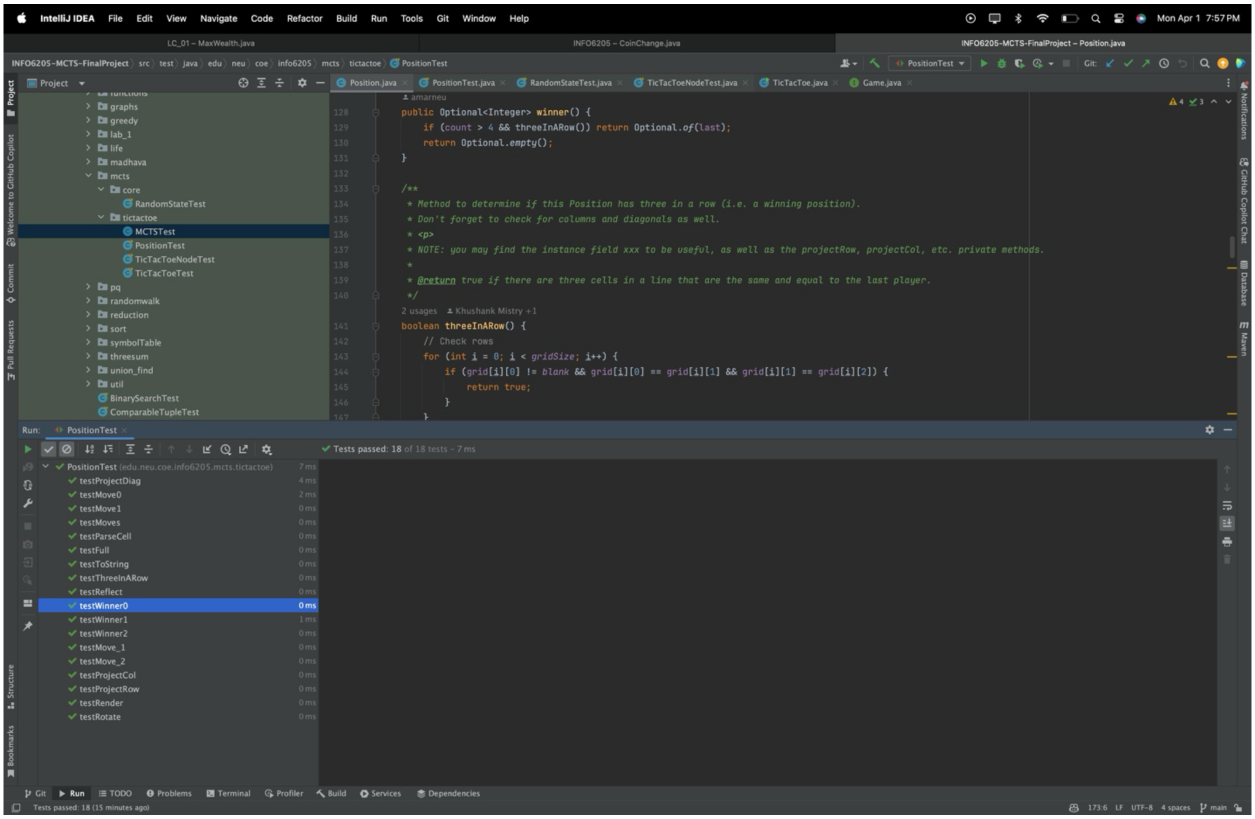
Run: TicTacToeNodeTest

✓ Tests passed: 6 of 6 tests – 4 ms

```
✓ TicTacToeNodeTest (edu.neu.coe.info6205.mcts.tictacto   4 ms
    ✓ addChild          1 ms
    ✓ state             2 ms
    ✓ white             0 ms
    ✓ backPropagate     1 ms
    ✓ children          0 ms
    ✓ winsAndPlayouts   0 ms
```

```
/Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

Tests passed: 6 (moments ago)



```java
public Optional<Integer> winner() {
    if (count > 4 && threeInARow()) return Optional.of(last);
    return Optional.empty();
}

/**
 * Method to determine if this Position has three in a row (i.e. a winning position).
 * Don't forget to check for columns and diagonals as well.
 * <p>
 * NOTE: you may find the instance field xxx to be useful, as well as the projectRow, projectCol, etc. private methods.
 *
 * @return true if there are three cells in a line that are the same and equal to the last player.
 */
boolean threeInARow() {
    // Check rows
    for (int i = 0; i < gridSize; i++) {
        if (grid[i][0] != blank && grid[i][0] == grid[i][1] && grid[i][1] == grid[i][2]) {
            return true;
        }
    }
```

```
✓ Tests passed: 18 of 18 tests – 7 ms

✓ PositionTest (edu.neu.coe.info6205.mcts.tictactoe)   7 ms
    ✓ testProjectDiag    4 ms
    ✓ testMove0          2 ms
    ✓ testMove1          0 ms
    ✓ testMoves          0 ms
    ✓ testParseCell      0 ms
    ✓ testFull           0 ms
    ✓ testToString       0 ms
    ✓ testThreeInARow    0 ms
    ✓ testReflect        0 ms
    ✓ testWinner0        0 ms
    ✓ testWinner1        1 ms
    ✓ testWinner2        0 ms
    ✓ testMove_1         0 ms
    ✓ testMove_2         0 ms
    ✓ testProjectCol     0 ms
    ✓ testProjectRow     0 ms
    ✓ testRender         0 ms
    ✓ testRotate         0 ms

Tests passed: 18 (15 minutes ago)
```

IntelliJ IDEA File Edit View Navigate Code Refactor Build Run Tools Git Window Help
Mon Apr 1 7:57 PM
INFO6205-MCTS-FinalProject – Position.java
Position.java PositionTest.java RandomStateTest.java TicTacToeNodeTest.java TicTacToe.java Game.java
graphs greedy lab_1 life madhava mcts core RandomStateTest tictactoe MCTSTest PositionTest TicTacToeNodeTest TicTacToeTest pq randomwalk reduction sort symbolTable threesum union_find util BinarySearchTest ComparableTupleTest
Run: PositionTest
Git Run TODO Problems Terminal Profiler Build Services Dependencies

Summary:

The provided code snippets from the TicTacToe game implementation focus on the Position class, which represents the game board as a 3x3 matrix. The class includes methods for parsing the board state from a string, making a move, listing all possible moves, reflecting, and rotating the board, and determining the winner. The implementation details for three critical methods were highlighted:

1. move(int player, int x, int y): This method updates the game board with a player's move, ensuring the move is valid and the position is not already occupied.
2. moves(int player): Generates a list of all possible moves for a player, considering the current state of the board.
3. threeInARow(): Checks for a winning condition by examining if there are three consecutive marks by the same player in any row, column, or diagonal.

These methods are crucial for the game's functionality, allowing for dynamic gameplay and determining the game's outcome based on players' actions.

Conclusion:

Implementing the specified parts of the TicTacToe game serves as a practical exercise in understanding the game's underlying mechanics and the application of Java generics in creating a flexible and reusable codebase. Through this milestone, learners are expected to gain insights into designing and coding game logic that can be applied not only to TicTacToe but potentially to other games with similar requirements. The focus on implementing functionality for player moves, evaluating possible moves, and determining winning conditions lays a solid foundation for further exploration and development of more complex game features or entirely new games.