

# INFO6205 Program Structures and Algorithms

Yu Lai NUID: 002640845  
Ting Guo NUID: 002834835  
Zenan Fan NUID: 002854067

GitHub: [https://github.com/laiyumi/INFO6205\\_final\\_project.git](https://github.com/laiyumi/INFO6205_final_project.git)

Spring 2024

## 1 Task: TicTacToe

## 2 Evidence

The screenshot shows the IntelliJ IDEA interface with the file `TicTacToe.java` open. The code implements a Tic Tac Toe game. In the `main` method, it creates a `TicTacToe` instance and prints the winner or draw. Below the code editor is the `Run` tool window, which shows the output of running the program. The output indicates a draw and displays the current board state:

```
Process finished with exit code 0
/Users/yulai/Library/Java/JavaVirtualMachines/openjdk-22/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=44284:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8
TicTacToe: winner is: 0
Current Position:
X 0 0
X 0 X
0 . X
```

Figure 1: Print out the result

```

public static void main(String[] args) {
    // NOTE the behavior of the game to be run will be based on the TicTacToe instance field: random.

    int[] testArrays = new int[]{500, 1000, 2000, 4000};
    Random random = new Random();

    for (int i = 0; i < testArrays.length; i++){

        // number of runs
        int n = testArrays[i];
        int player0win = 0;
        int player1win = 0;
        int opener = -1;

        for(int j = 0; j < n; j++){
            State<TicTacToe> state = new TicTacToe().runGame();
            opener = state.game().opener();
            if (state.winner().isPresent()){

                System.out.printf("#d TicTacToe: winner is: %d\n", i, state.winner().get());
                if(state.winner().get() == 1){
                    player1win++;
                } else{
                    player0win++;
                }
            }
            // display the result
            TicTacToeState ticTacToeState = (TicTacToeState) state;
            Position position = ticTacToeState.position();
            System.out.println("Result: \n" + position.render());
        }
    }
}

```

Process finished with exit code 0

Figure 2: Default setting

```

public static void main(String[] args) {
    // NOTE the behavior of the game to be run will be based on the TicTacToe instance field: random.

    int[] testArrays = new int[]{500, 1000, 2000, 4000};
    Random random = new Random();

    for (int i = 0; i < testArrays.length; i++){

        // number of runs
        int n = testArrays[i];
        int player0win = 0;
        int player1win = 0;
        int opener = -1;

        for(int j = 0; j < n; j++){
            State<TicTacToe> state = new TicTacToe(random).runGame();
            opener = state.game().opener();
            if (state.winner().isPresent()){

                System.out.printf("#d TicTacToe: winner is: %d\n", i, state.winner().get());
                if(state.winner().get() == 1){
                    player1win++;
                } else{
                    player0win++;
                }
            }
            // display the result
            TicTacToeState ticTacToeState = (TicTacToeState) state;
            Position position = ticTacToeState.position();
            System.out.println("Result: \n" + position.render());
        }
    }
}

```

Process finished with exit code 0

Figure 3: Random setting

The screenshot shows the Java code for TicTacToe. The code implements a game where the seed is set to -1. It runs 4000 games, each with 500 runs. The output shows that player 0 wins 0, player 1 wins 0, and draws 4000.

```

public class TicTacToe implements Game<TicTacToe> {
    /**
     * Main program to run a random TicTacToe game.
     *
     * @param args command-line arguments.
     */
    public static void main(String[] args) {
        // NOTE the behavior of the game to be run will be based on the TicTacToe instance field: random.

        int[] testArrays = new int[]{500, 1000, 2000, 4000};
        Random random = new Random();
        long seed = -1;

        for (int i = 0; i < testArrays.length; i++) {

            // number of runs
            int n = testArrays[i];
            int player0win = 0;
            int player1win = 0;
            int opener = -1;

            for (int j = 0; j < n; j++) {
                State<TicTacToe> state = new TicTacToe(seed).runGame();
                opener = state.game().opener();
                if (state.winner().isPresent()) {
                    System.out.printf("#d TicTacToe: winner is: %d\n", i, state.winner().get());
                    if (state.winner().get() == 1) {
                        player1win++;
                    } else {
                        player0win++;
                    }
                }
            }
        }
    }
}

```

Figure 4: Seed = -1

The screenshot shows the Java code for TicTacToe. The code implements a game where the seed is set to 0. It runs 4000 games, each with 500 runs. The output shows that player 0 wins 500, player 1 wins 500, and draws 3000.

```

public class TicTacToe implements Game<TicTacToe> {
    /**
     * Main program to run a random TicTacToe game.
     *
     * @param args command-line arguments.
     */
    public static void main(String[] args) {
        // NOTE the behavior of the game to be run will be based on the TicTacToe instance field: random.

        int[] testArrays = new int[]{500, 1000, 2000, 4000};
        Random random = new Random();
        long seed = 0;

        for (int i = 0; i < testArrays.length; i++) {

            // number of runs
            int n = testArrays[i];
            int player0win = 0;
            int player1win = 0;
            int opener = -1;

            for (int j = 0; j < n; j++) {
                State<TicTacToe> state = new TicTacToe(seed).runGame();
                opener = state.game().opener();
                if (state.winner().isPresent()) {
                    System.out.printf("#d TicTacToe: winner is: %d\n", i, state.winner().get());
                    if (state.winner().get() == 1) {
                        player1win++;
                    } else {
                        player0win++;
                    }
                }
            }
        }
    }
}

```

Figure 5: Seed = 0

The screenshot shows an IDE interface with the project 'INFO6205' open. In the center, the file 'TicTacToe.java' is displayed. The code implements a Tic Tac Toe game with a main method that runs 4000 games. It uses a seed value of 1. The run output window at the bottom shows the results of these games, indicating that Player 0 (opener) wins all 4000 games.

```

public class TicTacToe implements Game<TicTacToe> {
    /**
     * Main program to run a random TicTacToe game.
     *
     * @param args command-line arguments.
     */
    public static void main(String[] args) {
        // NOTE the behavior of the game to be run will be based on the TicTacToe instance field: random.

        int[] testArrays = new int[]{500, 1000, 2000, 4000};
        Random random = new Random();
        long seed = 1;

        for (int i = 0; i < testArrays.length; i++) {

            // number of runs
            int n = testArrays[i];
            int player0win = 0;
            int player1win = 0;
            int opener = -1;

            for (int j = 0; j < n; j++) {
                State<TicTacToe> state = new TicTacToe(seed).runGame();
                opener = state.game().opener();
                if (state.winner().isPresent()) {
                    System.out.printf("#d TicTacToe: winner is: %d\n", i, state.winner().get());
                    if (state.winner().get() == 1) {
                        player1win++;
                    } else {
                        player0win++;
                    }
                }
            }
        }
    }
}

```

Run TicTacToe

```

/Users/yuqiai/Library/Java/JavaVirtualMachines/openjdk-22/Contents/Home/bin/java ...
Total games: 500, opener: 1, player 0 wins 0, player 1 wins 500, draws 0
Total games: 1000, opener: 1, player 0 wins 0, player 1 wins 1000, draws 0
Total games: 2000, opener: 1, player 0 wins 0, player 1 wins 2000, draws 0
Total games: 4000, opener: 1, player 0 wins 0, player 1 wins 4000, draws 0
Process finished with exit code 0

```

Figure 6: Seed = 1

### 3 Observations and Conclusions

TicTacToe (default)					TicTacToe (Seed = -1)				
Runs	N = 500	N = 1000	N = 2000	N = 4000	Runs	N = 500	N = 1000	N = 2000	N = 4000
Player 0	107	410	803	1689	Player 0	0	0	0	0
Player 1 (opener)	393	415	1000	2311	Player 1 (opener)	500	1000	2000	4000
Draw	0	175	197	0	Draw	0	0	0	0

  

TicTacToe (Random)					TicTacToe (Seed = 0)				
Runs	N = 500	N = 1000	N = 2000	N = 4000	Runs	N = 500	N = 1000	N = 2000	N = 4000
Player 0	155	299	591	1149	Player 0	0	0	0	0
Player 1 (opener)	286	581	1134	2309	Player 1 (opener)	0	0	0	0
Draw	59	120	275	542	Draw	500	1000	2000	4000

  

TicTacToe (Seed = 1)				
Runs	N = 500	N = 1000	N = 2000	N = 4000
Player 0	0	0	0	0
Player 1 (opener)	500	1000	2000	4000
Draw	0	0	0	0

Figure 7: Test TicTacToe with random and seeds

From Figure 1, we can see that our code has achieved the correct conclusion. And our subsequent series of tests can also prove that my game implementation is in line with the actual situation from the data layer main. We can find from a large number of data experiments that the relationship between the two players is not one-sided, which is in line with the situation of random generation. But at the same time, the winning rate is always the same person, which is also in line with the characteristics of this game. Because tic-tac-toe can only take nine steps at most, one player can take five, and another can take four steps. So, the player who takes five steps will have a particular advantage, which is also reflected in the final winning rate. Furthermore, we observed that using seeds -1 and 1 creates deterministic conditions that always result in a win for the player who goes first. On the other hand, a seed of 0 results in all draws, indicating an implementation that ensures no victories.