Northeastern University

CSYE 7230 Software Engineering
Fall 2024

Project #1

Posted: 9/8/2024
~~**Due: 9/19/2024, 11:59pm**~~
Due: 9/22/2024, 11:59pm
Questions? Email me at a.alali@northeastern.edu

A. GitHub

Use your GitHub account, and if you don't have one, make one, then:

1. Create a new repository and call it **tic-tac-toe**. Make it private, and add me to it, my GitHub ID is aalali
2. Create a branch, call it **tic-tac-toe-feature**
3. Open a draft pull request, then commit your code as frequently as possible
4. Once finished, add me as a reviewer (due date is here), a reference on how to
5. I will review, and add comments, and then I will approve
6. You merge into the main, delete the feature branch

You might this reference helpful.

B. Object Oriented Modeling

The Scenario

Tic-tac-toe is a game for player vs. player (or player vs. a machine), X and O, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game. The game can rerun and count how many wins each player made.

Functional Requirements

1. Get a copy of the tic-tac-toe v0.9 game file (`tictactoe-v0.9.java`). Compile and execute the Java code using Java JDK 22. Play the game enough to understand it and read the code. tic-tac-toe v0.9 is a player vs. player type of a tic-tac-toe game.

2. Transform the game to a releasable state after advancing tic-tac-toe version 0.9. The new version runs more like the playtictactoe.org game. The visuals, display, interaction, and sound effects are to be ignored, focus should be more on functionalities.

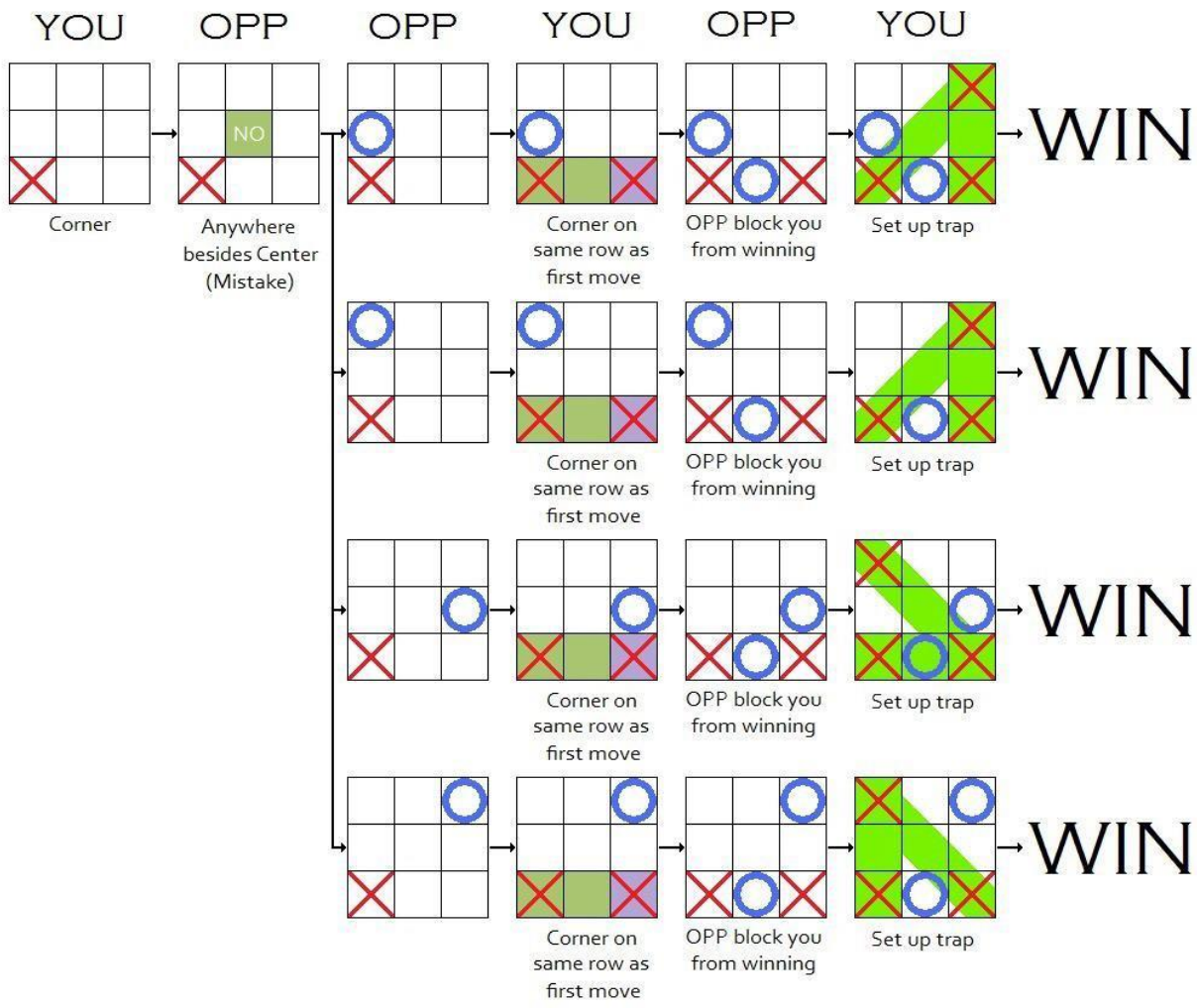
Functional requirements and features and bugs in tic-tac-toe v0.9 to be addressed in v1.0:

Issue #1 | Bug | A player can overwrite an already chosen cell in the grid

Issue #2 | Bug | Bad input (a character) will send the game into an infinite loop

Issue #3 | Bug | A tied game is not declared

Issue #4 | Feature | Transform the game from player vs. player (PvP) only to player vs. player and player vs. a machine (PvM). The user can have an option to select which type before a game starts. For PvM, there are two winning strategies: 1. The machine runs at least as smart as a random function on picking the next move 2. P is the first as X, while the machine is smart too as the second player. The machine would follow this strategy:



Issue #5 | **Feature** | Add wins and ties counters for each player (PvP or PvM) as in playtictactoe.org

Non-Functional Requirements

3. You are required to model/engineer the game by applying the object-oriented analysis and design (OOAD) methods and principles you have learned so far in class. The game should be decomposed into Java classes representing real-world domain objects. These objects encapsulate attributes and operations and interact with each other's by declaring relationships.

Hint: Conceptual objects to use can be a Square, Board, Player, TicTacToeGame, Engine, GameStart, GameResult, GameMark, RandomPick, GameScore, etc. These class/object names are only suggested names and not names that you must use!

4. Code compiles and runs with jdk.java.net/22

5. Write unit tests using junit.org/junit5, make sure I can understand your code reading only your unit tests!

6. Add good comments when necessary, and follow variable naming conventions in Java

7. Add your name as the author, in every file.

Artifacts to Submit

You need to submit the following:

8. One happy path use case scenario for (`scenarios.txt`)

9. A use case diagram that represents the game goals and interactions (`usecasediagram.png`)

10. A domain model/conceptual model of the game (`domainmodel.png`)

11. A class diagram of the modeled system (`classdiagram.png`)

12. A sequence diagram that describes the system based on the selected happy path (`sequencediagram.png`)

13. The code `*.java`

14. A `README.txt` file that explains how to compile, run your app, and how to run the tests

Your directory tree structure **should** look like this:

```
.
|-- project1
|   |   |-- tictactoe-v1.0
|   |   |   |-- docs
|   |   |   |   |-- scenarios.txt
|   |   |   |   |-- usecasediagram.png
|   |   |   |   |-- domainmodel.png
|   |   |   |   |-- classdiagram.png
|   |   |   |   |-- sequencediagram.png
|   |   |   |   |-- README.txt
|   |   |   |-- src
|   |   |   |   |-- *.java
```

Notes

1. Use draw.io, PlantUML, Mermaid, Lucidcharts or Microsoft Visio for UML diagraming or any UML tools you choose. Don't use hand drawing or something as simple as Microsoft Paint.
2. All diagrams have to be submitted as png or jpg type files. For documents, use txt type files. For code use .java type files.
3. Commit as frequently as possible, with adequate messages to show your progress
4. Use LLMs to help you understand, finish tasks, and write better code faster, Codium in vscode is free.

Grading

The grade will be broken down as follows:

Points	Mark
1, 2	(30/100)
3, 4, 5, 6, 7	(50/100)
8, 9, 10, 11, 12, 13, 14	(20/100)