

Grid-Based Game

So far we have learned about 1-dimensional arrays in Java. These are data structures with **one** index that can be used to represent a single row of related data.

Java also supports 2-dimensional arrays. These can be used to represent a grid (or table) of related data organized into **rows** and **columns**. For this reason, 2D arrays require **two** indices.



Before You Begin:

Before you attempt this assignment, you need to familiarize yourself with how 2D arrays actually work in Java! Here's some guidance:

- Mr. Rao will demonstrate a few 2D Array examples for you in class.
- Check out [3-3 Two-Dimensional Arrays](#) and the [Tic-Tac-Toe Example](#) on Vik-20.

Assignment Requirements:

1. **Pick a 2 player, grid-based game** of interest to you. For example:
 - Connect-4
 - Battleship
 - Treasure Hunt
 - Something else (just run it by me before you start)
 - **NOT** Tic-Tac-Toe (example code is given for this already)
2. **Create a Java program** to demonstrate your mastery of 2D arrays and Java class and method design. Your program must meet the following technical requirements:
 - a. Design and implement a **Java class to represent your game board**. Your class should use a 2D array to represent your game board. Demonstrate proper encapsulation techniques, including constructor(s), accessor(s), mutator(s), toString(), and any other methods you need to manipulate your board.
 - b. Draw a complete **UML diagram** for your game board class. Your UML diagram and Java code should reflect each other exactly.
 - c. Create another class containing your **main()** method. Your **main()** should instantiate your game board and include the **main game loop** for your program.

- d. You should create additional “helper” methods that **main()** will call to run the game. Have a look at the [Tic-Tac-Toe Example](#) on Vik-20.
 - e. Keep all of your methods focused and under 30-40 lines in length.
 - f. Minimize scope and avoid defining static class variables anywhere in your code, use parameters and return values to effectively communicate between your functions.
 - g. Your game should be designed such that the user plays against the computer. For a higher level of difficulty, rather than having the computer make completely random moves, consider having the computer make calculated, strategic moves instead so that it is harder to beat.
3. **Follow these style rules** for maximum readability:
- a. Include a **Javadoc** style `/** multi-line */` comment at the top of your code with your name, date, and brief description of your project.
 - b. Include a **Javadoc** style `/** multi-line */` comment for **every** method in your program explaining its: (1) purpose, (2) parameters (if any), and (3) return value (if any).
 - c. Include `//` single-line comments to describe tricky code or significant groups of related Java statements.
 - d. Make effective use of 4-space indents, and blank lines between functions and groups of related Java statements.
 - e. Use meaningful, descriptive variable and function names following camelCase.
4. **Reflect on your game.** Complete the reflection questions and rubric below.
5. **Demonstrate your game.** Once you have met all of requirements above, give Mr. Rao a demo of your game and discuss your self-assessment.

Reflecting on Your Assignment

- A. What aspects of the project were the most challenging for you, and why?

- B. If you did not meet all the requirements, which ones do you need to improve on?





- C. What part of your project are you most proud of?

Reflecting on Your Assignment

Name: _____

Topic: _____

Considering the requirements, how would you rate your project? (✓ Check one)

Criteria	 1 = Poor	 2 = Fair	 3 = Good	 4 = Great
Technical Requirements (2a,b,c,d,e,f,g)	Too many technical requirements missing / incomplete.	Improvement needed on 3 or more technical requirements.	Most code requirements met, but need to improve on 1 or 2 of them.	All code requirements met! Exemplary Java code.
Coding Style Rules (3a, b, c, d, e)	Poor style, 3 or more style rules missing or ignored.	Missing or incomplete Javadocs, room to improve on 2 style rules.	Good Javadoc style, but room to improve on 1 style rule.	Perfect Javadoc style, and great attention to detail!
Application Complexity	Trivial game, limited effort demonstrated.	Application meets most requirements, but buggy. More testing needed!	Strong effort, project meets requirements, computer moves are "random"	Challenging project, computer moves are "strategic"