

Project 1: Minesweeper

DUE: September 8th at 11:59pm
Extra Credit Available for Early Submissions!

Basic Procedures

You must:

- Fill out a readme.txt file with your information (goes in your user folder, an example readme.txt file is provided).
- Have a style (indentation, good variable names, etc.) and pass the provided style checker (see P0).
- Comment your code well in JavaDoc style and pass the provided JavaDoc checker (see P0).
- Have code that compiles in your user directory without errors or warnings.
- For methods that come with a big-O requirement (check the provided template Java files for details), make sure your implementation meet the requirement.
- Implement all required methods to match the expected behavior as described in the given template files.
- Have code that runs (see detailed commands below).

You may:

- Add additional methods and class/instance variables, however they **must be private**.

You may NOT:

- Make your program part of a package.
- Add additional public methods or public class/instance variables, remember that local variables are not the same as class/instance variables!
- Use any built in Java Collections Framework classes anywhere in your program (e.g. no ArrayList, LinkedList, HashSet, etc.).
- Declare/use any arrays anywhere in your program (except the **storage** field provided in the **DynArr310**).
 - When you need to expand or shrink the provided data field, it is fine to declare/use a "replacement" array.
- Alter any method signatures defined in this document of the template code. Note: “throws” is part of the method signature in Java, don’t add/remove these.
- Alter provided methods or classes that are complete (e.g. **MineGUI**).
- Add any additional import statements (or use the “fully qualified name” to get around adding import statements).
- Add any additional libraries/packages which require downloading from the internet.

Setup

- Download the **p1.zip** and unzip it. This will create a folder **section-yourGMUUserName-p1**;
- Rename the folder replacing **section** with the lecture section you are in;
- Rename the folder replacing **yourGMUUserName** with the first part of your GMU email address;
- After renaming, your folder should be named something like: **000-yzhong-p1**.
- Complete the **readme.txt** file (an example file is included: **exampleReadmeFile.txt**).

Submission Instructions

- Make a backup copy of your user folder!
- Remove all test files, jar files, class files, etc. You should just submit your java files and your readme.txt
- Zip your user folder (not just the files) and name the zip **section-username-p1.zip** (no other type of archive) following the same rules for **section** and **username** as described above. For example:
 - **000-yzhong-p1.zip --> 000-yzhong-p1 --> JavaFile1.java
 JavaFile2.java
 JavaFile3.java ...**
- Submit to blackboard.

Grading Rubric

Due to the complexity of this assignment, an accompanying grading rubric pdf has been included with this assignment. Please refer to this document for a complete explanation of the grading, including extra credit for early submissions.

Overview

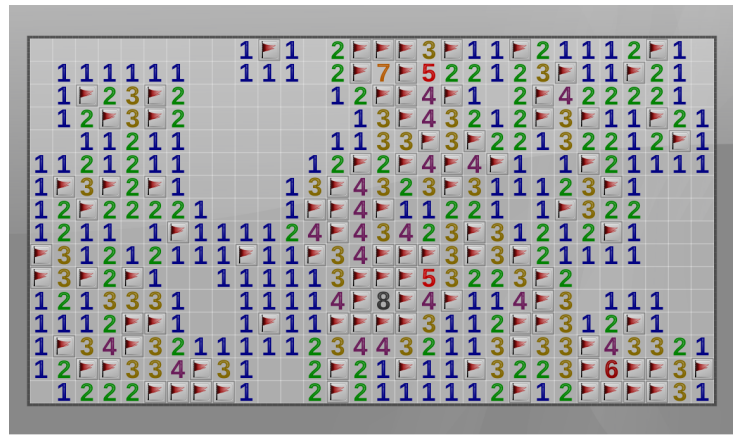
There are three major components to this project:

1. Implementing one of the most fundamental data structures in computer science (the dynamic array list).
2. Using this data structure to implement a larger program.
3. Practicing many fundamental skills learned in prior programming courses including generic classes.

The end product of this project will be a program with a graphical user interface to play a Minesweeper game.

(Picture shown to the right, By Brandenads - Own work, CC BY-SA 4.0,

<https://commons.wikimedia.org/w/index.php?curid=112673419>). Minesweeper is a game with a 2-D grid where a subset of the cells of the grid having "mines" hidden underneath them. There is either 0 or 1 mine underneath each cell. To play the game, the user needs to decide which cells do not contain a mine and click to expose them. If you click a cell with a mine underneath it, however, the mine would explode and you lose the game. Hence the goal of a Minesweeper game is to uncover all cells that do not contain mines without getting any of the mines exploded. In this project, we will use the dynamic array list to implement the internal storage of the grid and support the gameplay operations as described below.



Board Setting. Minesweeper starts with a 2-D grid with N rows, M columns, and N*M individual cells. We will support pre-defined levels with set sizes and predetermined number of mines, as listed in the table to the right. In our internal storage, we assign a row index (starting from 0 from top down) and a column index (starting from 0 from left to right) to each cell. The picture below shows the indexes we use for an example 5 by 5 board (level Tiny).

Table 1: Levels of the game.

| Level | Row Count | Column Count | Mine Count |
|---------------|-----------|--------------|------------|
| Tiny | 5 | 5 | 3 |
| Easy | 9 | 9 | 10 |
| Medium | 16 | 16 | 40 |
| Hard | 16 | 30 | 99 |

| | | | | | | | |
|-------------------|----------|----------------------|----------|----------|----------|----------|----------|
| | | Column index: | 0 | 1 | 2 | 3 | 4 |
| Row index: | 0 | | | | | | |
| | 1 | | | | | | |
| | 2 | | | | | | |
| | 3 | | | | | | |
| | 4 | | | | | | |

Game Status. Initially, the GUI shows an empty board and all cells are hidden, as shown by Example 1 to the right. As shown, the bottom of the GUI window would be updated to show the current game status. The provided GUI will automatically update those, given that your code is correctly implemented to maintain the status.

- **INIT:** this indicates a new game has just started and the user has not performed any operations yet.
- **IN_GAME:** once the user performs the first move (either a click or a flag), the status is changed to IN_GAME.
- **EXPLODED:** this is triggered if any cell with a mine is clicked.
- **SOLVED:** this status is reached when all cells without a mine have been opened before any explosion.
- **MINES:** this is basically a hint to the users and always reports the number of mines to be flagged. **Note:** It does not check whether a flag is correct or not. It simply reports the difference between total number of mines and the number of flagged cells.



Example 1: Initial board for level Tiny.

Cell States. Cells in the game board have four states: **hidden**, **exposed**, **flagged**, or **exploded**.







- A **hidden** cell is blank, which can be click to expose whether it has a mine or not. The user can also flag the cell if they decide it has a mine underneath the cell.

TL;DR You need to implement basic data structures to support the implementation of a Minesweeper game.

- An **exposed** cell has already been clicked and confirmed it does not contain a mine. Its state cannot be changed.
- A **flagged** cell has been marked by the user as a candidate with a mine underneath it. If the user changes their mind, they can unflag the cell and it goes back to the hidden state.
- When the user clicks on a hidden cell with a mine, it will **explode** and end the game.

Gameplay operations. The user can perform three kinds of operations to play a Minesweeper game:

- **Click** (open) a hidden cell. It will open the cell to show whether it has a mine or not. If it has a mine, the mine gets exploded and the game ends with a loss. Otherwise, the cell will display information regarding the number of mines in its neighboring cells (left, right, top, down, as well as diagonally adjacent neighbors).
 - If there is at least one mine in its neighbors, a number (1-8) will be displayed on the cell. This is the critical information for the user to deduce which cells are safe to open.
 - If the cell is mine-free and not adjacent to any mines, we call it a zero-count cell. Clicking on a zero-count cell will trigger a sequence of flipping and automatically open all zero-count cells that are connected to the original cell, as well as all cells that are orthogonally or diagonally adjacent to those zero-count cells. Check examples below.
- **Flag** a hidden cell. It will mark the cell to indicate there is a mine underneath it. A flagged cell cannot be clicked but one can choose to unflag it (see below) if they decide the flag is incorrect.
- **Unflag** a flagged cell. It will change the cell back to hidden.

| Original Board | Operations | Updated Board | Notes |
|---|---|--|---|
|  | Click cell at (3,3) |  | <ul style="list-style-type: none"> • The click opens cell at (3,3) since it has no mine. • It displays a number 3, which means there are three mines hidden beneath the neighbor cells. • After the first click, game status is changed from INIT to IN_GAME. |
|  | Click cell at (0,0) |  | <ul style="list-style-type: none"> • Cell (0,0) is a zero-count cell. It shows as a blank cell after click. • A group of additional cells will also open by that click, including: <ul style="list-style-type: none"> • zero-count cells connected to the original cell (0,0), e.g. (0,1) • neighbor cells of those zero-count cells, e.g. (1,2) |
|  | Flag cell at (2,3); Flag cell at (2,4) |  | <ul style="list-style-type: none"> • Flagged cells are marked with 'F'. • Number of remaining mines updated. No checking whether the flagging is correct or not. |



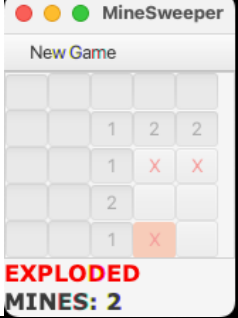


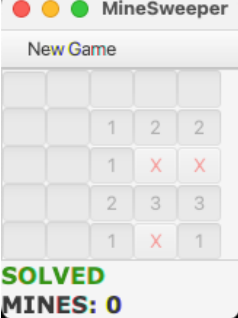
| | | | |
|---|---|--|---|
|  | Unflag cell at (2,4) |  | <ul style="list-style-type: none"> Unflag a cell would make it go back to be hidden. Further clicking or flagged can be applied to the cell. Number of mines at the bottom will be updated. |
|  | Click cell at (4,3) |  | <ul style="list-style-type: none"> Clicking on a cell with a mine will trigger a mine explosion. Game ends with status reported as EXPLODED. All mine cells will be exposed with the one exploded highlighted. |
|  | Click cell at (3,4); Click cell at (4,4) |  | <ul style="list-style-type: none"> One must open all non-mine cells to solve the board and win the game. Game status shows as SOLVED once you win the game. All mine cells will be exposed. |
|  | Click cell at (4,4) |  | <ul style="list-style-type: none"> Flagging mine cells are not required. All mine cells will be exposed once the board is solved regardless whether they have been flagged or not. |

Table 2: Example operations and their effects. (Level: Tiny)

TL;DR Minesweeper GUI provided but you need to implement gameplay operations.

Implementation/Classes

This project will be built using a number of classes representing the generic dynamic array, the generic 2-D grid, the cell, and the Minesweeper game as we described in the previous section. Here we provide a description of these classes. Template files are provided for each class in the project package and these contain further comments and additional details. You must follow the instructions included in those files.

- DynArr310 (DynArr310.java):** The implementation of a dynamic array. You will implement this class as a generic class to practice that concept.
- DynGrid310 (DynGrid310.java):** The implementation of a 2-dimensional grid. It is also a generic class and organized as a dynamic array of dynamic arrays.
- Cell (Cell.java):** The class representing one cell in the game board. player record. This class is provided to you and you should NOT change the file. Check the file to see how to maintain/check the status of a cell.

- **MineSweeper (MineSweeper.java):** The implementation of a Minesweeper game. This class is partially implemented. Do NOT change the provided code in the file. You will need to complete the missing methods and add JavaDoc.
- **MineGUI (MineGUI.java):** A graphical user interface for the Minesweeper game. This class is provided to you and you should NOT change the file.

Requirements

An overview of the requirements is listed below, please see the grading rubric for more details.

- **Implementing the classes** - You will need to implement required classes and fill the provided template files.
- **JavaDocs** - You are required to write JavaDoc comments for all the required classes and methods.
- **Big-O** - Template files provided to you contains instructions on the REQUIRED Big-O runtime for many methods. Your implementation of those methods should NOT have a higher Big-O.

How To Handle a Multi-Week Project

While this project is given to you to work on for about two weeks, you are unlikely to be able to complete everything in one weekend. We recommend the following schedule:

- Step 1 (Prepare): by 08/28
 - Go over Project 1 with a fine-toothed comb.
 - Read about Dynamic Array List (Ch15 of textbook).
 - Get familiar with Minesweeper game.
- Step 2 (**DynArr310**, **DynGrid310**): Before the second weekend (08/29-09/01)
 - Implement and test methods of **DynArr310**.
 - Implement and test methods of **DynGrid310**.
- Step 3 (**MineSweeper**): Second weekend (09/02-09/04)
 - Complete implementation and testing of **MineSweeper**.
- Step 4 (**Wrapping-up**): Last week (09/05-09/08)
 - Additional testing, debugging, get additional help.
 - ☺ Also, notice that if you get it done early in the week, you can get extra credit! Check our grading rubric PDF for details.

Testing

The main methods provided in the template files contain useful example code to test your project as you work. You can use command like "java DynArr310" or "java MineSweeper" to run the testing defined in `main()`.

- **Note:** passing all yays does NOT guarantee 100% on the project! Those are only examples for you to start testing and they only cover limited cases. Make sure you add many more tests in your development. You could edit `main()` to perform additional testing.
- Make sure you test all required methods of classes – even though not all of them are used in the end-product game, they will be checked in grading. Check the rubric PDF for a detailed breakdown of points.
- JUnit test cases will not be provided for this project, but feel free to create JUnit tests for yourself. A part of your grade *will* be based on automatic grading using test cases made from the specifications provided.

The provided **MineGUI** can be run to play the Minesweeper game.

- The GUI includes a menu that can be used to switch levels and create new game boards, including customized board sizes. It uses a fixed seed for random number generations. Hence it would be straightforward to repeat the same game if you re-run the program.
- Play different games / scenarios to test your MineSweeper implementation. Do not change the provided class to make the game work. In grading, we will test MineSweeper directly without relying on the GUI.
- You need to install and set up JavaFX to compile and run **MineGUI**. Check below for details.

Additional Installation & Settings for GUI

The provided **MineGUI** is implemented using JavaFX. In order to compile and run **MineGUI.java**, the following steps need to be followed:

1. Download JavaFX runtime [17.0.8 \(SDK version\)](https://gluonhq.com/products/javafx/) for your machine and operating system from: <https://gluonhq.com/products/javafx/>. Note that they come with a minimum requirement on JDK. You will need to ensure your Java version is at least **Java 11**.
 - You will need to pick the version that matches your OS and processor.
 - For Windows, options are x86 (for 32-bit processors) and x64 (for 64-bit processors).
 - For Mac, options are x64 (for Intel processors) and aarch64 (for apple processors).
 - For Linux, options are x64 (for Intel processors), arm32 (for 32-bit ARM processors), and aarch64 (for 64-bit ARM processors).
2. Unzip the downloaded file to a desired location outside of your P1 working folder. This is your **JavaFX** folder.
3. Add an environment variable pointing to the **lib** folder inside the **JavaFX** folder. Use the absolute path to your JavaFX folder to make the following steps easier.
 - For Linux/Mac, use command:
`export PATH_TO_FX=path/to/JavaFX/lib`
 - Example: on a Mac computer with **JavaFX** installed at `/Users/yzhong/310/javafx-sdk-17.0.8/`, I would type:
`export PATH_TO_FX=/Users/yzhong/310/javafx-sdk-17.0.8/lib`
 - For Windows, use command:
`set PATH_TO_FX="path\to\JavaFX\lib"`
 - Example: on a Windows computer with **JavaFX** installed at `c:\cs310\javafx-sdk-17.0.8`, I would type:
`set PATH_TO_FX="c:\cs310\javafx-sdk-17.0.8\lib"`
 - **NOTE:** Setting a path variable on the command line only works for that one command prompt session. For any new or additional prompts you open, you will need to keep using the "set" or "export" command. If you want to, you can also set the environmental variable **JavaFX** permanently by changing your system settings.
4. Compile **MineGUI.java** by specifying modules included.
 - For Linux/Mac, use command:
`javac --module-path $PATH_TO_FX --add-modules javafx.controls MineGUI.java`
 - For Windows, use command:
`javac --module-path %PATH_TO_FX% --add-modules javafx.controls MineGUI.java`
5. Run the generated **MineGUI** class in a similar way.
 - For Linux/Mac, use command:
`java --module-path $PATH_TO_FX --add-modules javafx.controls MineGUI`
 - For Windows, use command:
`java --module-path %PATH_TO_FX% --add-modules javafx.controls MineGUI`

NOTE:

- Check <https://openjfx.io/openjfx-docs/#install-javafx> for other details.
- These steps are only needed for **MineGUI**. JavaFX is not required to complete the code required for **DynArr310**, **DynGrid310**, or **MineSweeper**.