Com S 227
Spring 2023
Assignment 3
300 points

Due Date: Monday, April 7, 11:59 pm (midnight)
5% bonus for submitting 1 day early (by 11:59 pm April 6)
10% penalty for submitting 1 day late (by 11:59 pm April 8)
No submissions accepted after April 8, 11:59 pm

This assignment is to be done on your own. See the Academic Integrity policy in the syllabus, for details.

You will not be able to submit your work unless you have completed the *Academic Dishonesty policy questionnaire* on the Assignments page on Canvas. Please do this right away.

If you need help, try office hours, the schedule is on Canvas. Lots of help is also available through the Piazza discussions. Please start the assignment as soon as possible to get your questions answered right away. It is physically impossible for the staff to provide individual help to everyone the afternoon that the assignment is due!

This is a "regular" assignment so we are going to read your code. Your score will be based partly on the specchecker's functional tests and partly on the grader's assessment of the quality of your code. See the "More about grading" section.

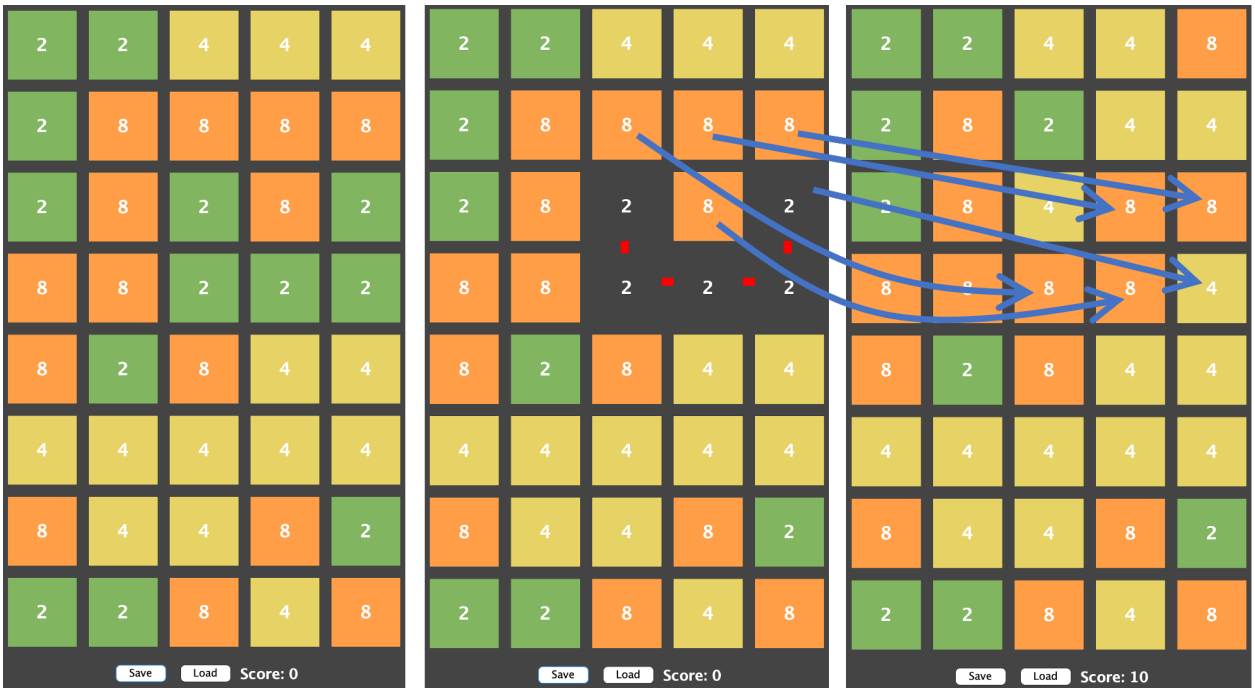
Contents

Introduction

The purpose of this assignment is to give you some practice writing loops, using arrays and lists, and, most importantly, to get some experience putting together a working application involving several interacting Java classes.

There are three classes for you to implement: **`ConnectGame, Grid, and GameFileUtil`**. As always, your primary responsibility is to implement these classes according to the specification and test them carefully. The three classes can be used, along with some other components, to create an implementation of our version of the popular puzzle game 2248.

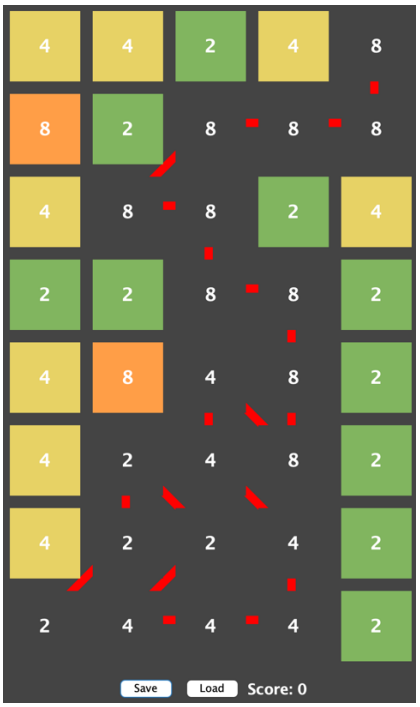
The game is played on a 2-dimensional grid of tiles. The player connects together a sequence of tiles, resulting in those tiles being eliminated from the board. Points are awarded for all tiles eliminated and new tiles drop down from the top of the grid to replace the missing tiles. A sequence of tiles must always start with two identical tiles followed by additional identical tiles or tiles in increasing powers of two. For example, 2 can connect to 4, which can connect to 8, which can connect to 16, and so on. The last tile in the sequence is replaced with a tile that is a power of 2 larger in value.



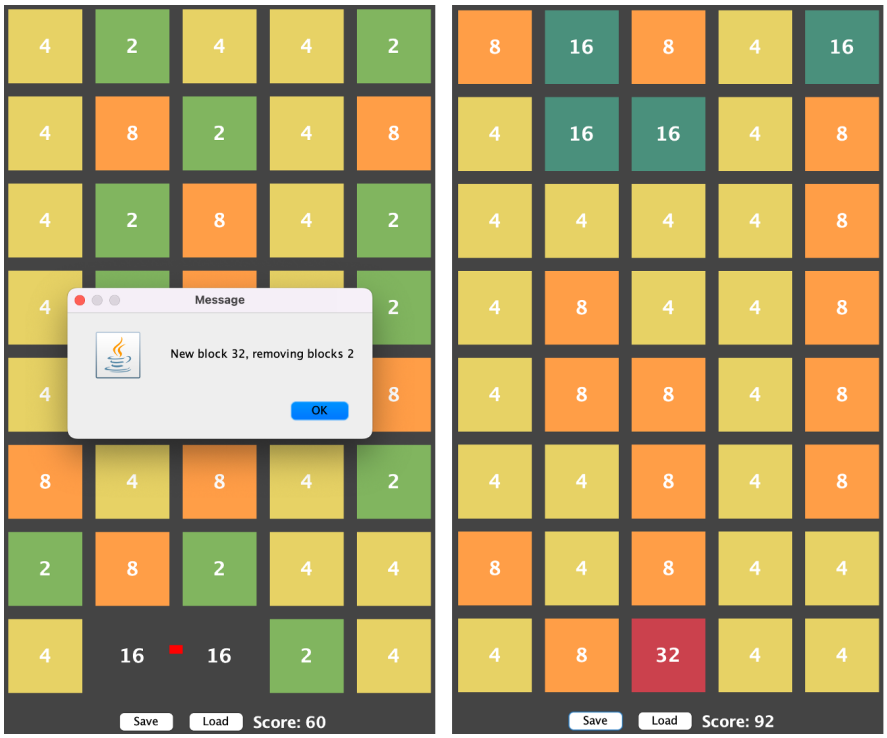
The figure above shows the game at three different times. On the left is the starting state. In the middle, the user has selected a sequence 2s starting from top left and ending top right. On the right, the last 2 has been converted into a 4, the existing 4s and 8s have dropped down to replace

the eliminated tiles, and new random tiles have dropped from above. Finally, the score has been updated to 10 because the sequence contained five 2s (5 x 2 = 10).

The figure below shows an example of a much longer sequence starting with the 2 at the bottom left and ending with the 8 at the top right.



At the start of the game the minimum tile is 2 and the maximum is 16. Tiles are generated randomly between the minimum inclusive and the maximum exclusive (in other words, the random tiles include 2 to 8). When the player crates a tile larger than the maximum, both the minimum and maximum are increased by a power of 2 and all tiles lower than the new minimum are eliminated from the grid. The tiles drop down from above to take the place of the eliminated ones. The figure below shows the user creating the first 32 tiles which results in all 2 tiles being eliminated from the grid.



The three classes you implement will provide the "backend" or core logic for the game. In the interest of having some fun with it, we will provide you with code for a GUI (graphical user interface), based on the Java Swing libraries.

The sample code includes a documented skeleton for the classes you are to create in the package **`hw3`**. The additional code is in the packages **`ui`** and **`api`**. The **`ui`** package is the code for the GUI, described in more detail in a later section. The **`api`** package contains some relatively boring types for representing data in the game. There are a few examples of test cases for you to start with located in the default package, along with a simple text-based user interface.

You should not modify the code in the **`ui or api`** packages.

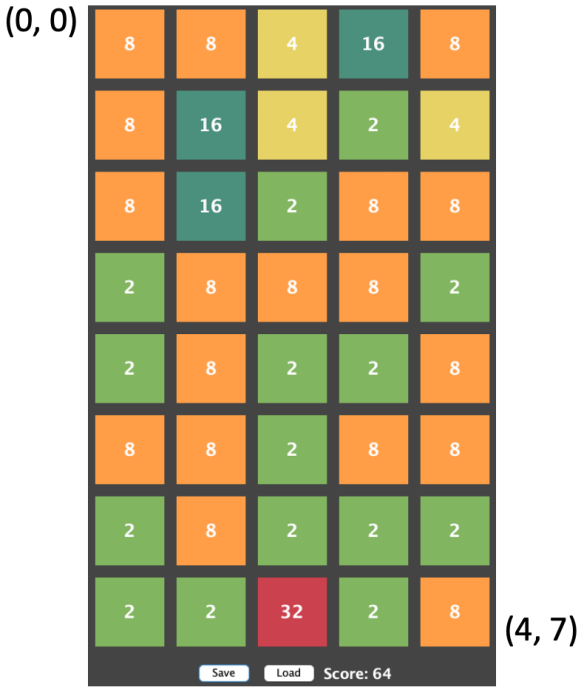
Specification

The complete specification for this assignment includes

- this pdf,
- the online Javadoc, and
- any "official" clarifications posted on Canvas and/or Piazza

Overview of the Grid class

A `Grid` object maintains a 2D array of tiles. The grid has a width and a height. Positions on the grid are specified as (x, y). In graphical representation, position (0, 0) is at the top left corner of the window.



Overview of the ConnectGame class

The `ConnectGame` class models the game and is called by the UI. For example, the UI calls `randomzeTiles()` at the start of the game to generate a random grid of tiles.

The main interaction with the user happens when the user clicks on a tile, moves the mouse over additional tiles, and then clicks on a tile to finalize the selection. These mouse events result in calls to methods. The first click calls `tryFirstSelect()`. As the mouse moves over additional blocks calls to `tryContinueSelect()` are made. A second mouse click results in calling `tryFinishSelection()`.

Some of the methods are intended to be called by other methods in the class. For example, `upgradeLastSelected()`, `dropSelected()`, and `dropLevel()` are intended to be called by the previously described methods when a selection has been finished.

Overview of the GameFileUtil class

The **`GameFileUtil`** class contains two static methods used for saving and loading the game.

The GUI

There is also a graphical UI in the **`ui`** package. The GUI is built on the Java Swing libraries. This code is complex and specialized, and is somewhat outside the scope of the course. You are not expected to be able to read and understand it. However, you might be interested in exploring how it works at some point. In particular, it is sometimes helpful to look at how the UI is calling the methods of the classes you are writing.

The main method is in **`ui.GameMain`**. You can try running it, and you'll see the initial window, but until you start implementing the required classes you'll just get errors. All that the main class does is to initialize the components and start up the UI machinery. The class **`GamePanel`** contains most of the UI code and defines the "main" panel.

Testing and the SpecChecker

As always, you should try to work incrementally and write tests for your code as you develop it.

Do not rely on the UI code for testing! Trying to test your code using a UI is very slow, unreliable, and generally frustrating. The code for the UI itself is more complex than the code you are implementing, and it is not guaranteed to be free of bugs***. In particular, when we grade your work we are NOT going to run the UI, we are going to test that each method works according to its specification.***

We will provide a basic SpecChecker, but **it will not perform any functional tests of your code**. At this point in the course, you are expected to be able to read the specifications, ask questions when things require clarification, and write your own unit tests.

Since the test code is not a required part of this assignment and does not need to be turned in, **you are welcome to post your test code on Piazza for others to check, use and discuss.**

The SpecChecker will verify the class names and packages, the public method names and return types, and the types of the parameters. If your class structure conforms to the spec, you should see a message similar to this in the console output:

```
x out of x tests pass.
```

This SpecChecker will also offer to create a zip file for you that will package up the two required classes. Remember that your instance variables should always be declared **`private`**, and if you

want to add any additional "helper" methods that are not specified, they must be declared `private` as well.

Importing the sample code

The sample code includes a complete skeleton of the three classes you are writing. It is distributed as a complete Eclipse project that you can import. It should compile without errors out of the box.

1. Download the zip file. You don't need to unzip it.
2. In Eclipse, go to File -> Import -> General -> Existing Projects into Workspace, click Next.
3. Click the radio button for "Select archive file".
4. Browse to the zip file you downloaded and click Finish.

If you have an older version of Java or if for some reason you have problems with this process, or if the project does not build correctly, you can construct the project manually as follows:

1. Extract the zip file containing the sample code (*Note: on Windows, you have to right click and select "Extract All" – it will not work to just double-click it*)
2. In Windows Explorer or Finder, browse to the src directory of the zip file contents
3. Create a new empty project in Eclipse
4. In the Eclipse Package Explorer, navigate to the src folder of the new project.
5. Drag the `hw3`, `ui`, and `api` folders from Explorer/Finder into the `src` folder in Eclipse.
6. Copy the remaining top-level files (SimpleTest.java, etc) into the `src` folder.

Getting started

At this point we expect that you know how to study the documentation for a class, write test cases, and develop your code incrementally to meet the specification, so this getting started section will not be quite as detailed as for the previous assignments.

You can find the example test cases below in the default package of the sample code. *Don't try to copy/paste from the pdf.*

Please remember that these are just examples to get you started and you will need to write many, many more tests to be sure your code is working correctly. (You can expect that the functional tests we run when grading your work will have something like 100 test cases, for example.)

Don't rely on the GUI for testing your code. Doing so is difficult and frustrating because the GUI itself is so complex. Rely on simple, focused test cases that you can easily run in the debugger.

A good starting point is the Grid class. This class should keep a 2D array of Tile objects. The methods are simple getters and setters.

```
// Example use of Grid
Grid grid = new Grid(2, 2);
grid.setTile(new Tile(1), 0, 0);
grid.setTile(new Tile(2), 1, 0);
grid.setTile(new Tile(1), 0, 1);
grid.setTile(new Tile(1), 1, 1);
System.out.println("Grid tests");
System.out.println("Width: " + grid.getWidth());
System.out.println("Height: " + grid.getHeight());
System.out.println("Tile: " + grid.getTile(0, 1));
System.out.println(grid);
System.out.println();
```

The `ConnectGame` class models the game and its methods are typically called by the UI. Write simple tests to take place of the UI. The main interaction with the user happens when the user clicks on a tile, moves the mouse over additional tiles, and then clicks on a tile to finalize the selection. These mouse events result in calls to methods. The first click calls `tryFirstSelect()`. As the mouse moves over additional blocks calls to `tryContinueSelect()` are made. A second mouse click results in calling `tryFinishSelection()`. Some of the methods are intended to be called by other methods in the class. For example, `upgradeLastSelected()`, `dropSelected()`, and `dropLevel()` are intended to be called by the previously described methods when a selection has been finished.

```
// Example use of ConnectGame
ConnectGame game = new ConnectGame(2, 2, 1, 4, new Random(0));
GameConsole gc = new GameConsole();
game.setListeners(gc, gc);
game.setGrid(grid);

System.out.println("Random tests");
Tile randomTile = game.getRandomTile();
// the test will always produce the same result because Random has
// been given a seed
System.out.println(randomTile);

game.radomizeTiles();
System.out.println(grid);
System.out.println();

System.out.println("Selection tests");
System.out.println(game.isAdjacent(grid.getTile(0, 0), grid.getTile(1,
1))); // expected true
System.out.println(game.tryFirstSelect(0, 0)); // expected true
System.out.println(game.tryFirstSelect(0, 0)); // expected false

System.out.println(Arrays.toString(game.getSelectedAsArray())); //
(0,0,4,true)
game.tryContinueSelect(1, 0);
System.out.println(Arrays.toString(game.getSelectedAsArray())); //
(0,0,4,true)
game.tryContinueSelect(0, 1);
```

```
      System.out.println(Arrays.toString(game.getSelectedAsArray())); //
(0,0,4,true), (0,1,4,true)
      game.tryContinueSelect(1, 0);
      System.out.println(Arrays.toString(game.getSelectedAsArray())); //
(0,0,4,true), (0,1,4,true), (1,0,8,true)

      game.tryFinishSelection(0, 1);
      System.out.println(Arrays.toString(game.getSelectedAsArray())); //
(0,0,4,true), (0,1,4,true), (1,0,8,true)
      game.tryFinishSelection(1, 0);
      System.out.println(Arrays.toString(game.getSelectedAsArray())); // []
      System.out.println(game.getScore()); // 16

      // [   (0,0,8,false),  (1,0,16,false)]
      // [   (0,1,8,false),   (1,1,8,false)]
      System.out.println(game.getGrid());
      System.out.println();

      game.dropLevel(3);
      // [   (0,0,2,false),   (1,0,8,false)]
      // [   (0,1,2,false),  (1,1,16,false)]
      System.out.println(game.getGrid());
      System.out.println();
```

The final class is `GameFileUtil` class. Use a text editor to open the saved file and check that it is in the expected format.

```
      System.out.println("Save and load tests");
      GameFileUtil.save("game.txt", game);
      GameFileUtil.load("game.txt", game);
      System.out.println(game.getGrid());
```

The SpecChecker

You can find the SpecChecker on Canvas. Import and run the SpecChecker just as you practiced in Lab 1. It will run a number of functional tests and then bring up a dialog offering to create a zip file to submit. Remember that error messages will appear in the *console* output. There are many test cases so there may be an overwhelming number of error messages. ***Always start***

reading the errors at the top and make incremental corrections in the code to fix them. When you are happy with your results, click "Yes" at the dialog to create the zip file. See the document “SpecChecker HOWTO”, if you are not sure what to do.

Style and documentation

Roughly 10% of the points will be for documentation and code style. The general guidelines are the same as in homework 2. However, since the skeleton code has the public methods fully documented, there is not quite as much to do. Remember the following:

- You must add an @author tag with your name to the javadoc at the top of each of the classes you write (in this case the classes in the hw3 package).
- You must javadoc each instance variable and helper method that you add. Anything you add must be **`private`**.
- Since the code includes some potentially tricky loops to write, **you ARE expected to add internal (//-style) comments, where appropriate**, to explain what you are doing inside the longer methods. A good rule of thumb is: if you had to think for a few minutes to figure out how to do something, you should probably include a comment explaining how it works. Internal comments always *precede* the code they describe and are indented to the same level.
- Keep your formatting clean and consistent.
- Avoid redundant instance variables
- Accessor methods should not modify instance variables

If you have questions

For questions, please see the Piazza Q & A pages and click on the folder **`hw3`**. If you don't find your question answered, then create a new post with your question. Try to state the question or topic clearly in the title of your post, and attach the tag **`hw3`**. *But remember, do not post any source code for the classes that are to be turned in.* It is fine to post source code for general Java examples that are not being turned in, and **for this assignment you are welcome to post and discuss test code**. (In the Piazza editor, use the button labeled "code" to have the editor keep your code formatting. You can also use "pre" for short code snippets.)

If you have a question that absolutely cannot be asked without showing part of your source code, change the visibility of the post to "private" so that only the instructors and TAs can see it. Be sure you have stated a specific question; vague requests of the form "read all my code and tell me what's wrong with it" will generally be ignored.

Of course, the instructors and TAs are always available to help you. See the Office Hours section of the syllabus to find a time that is convenient for you. We do our best to answer every question carefully, short of actually writing your code for you, but it would be unfair for the staff to fully review your assignment in detail before it is turned in.

Any announcements from the that are labeled "Official Clarification" are considered to be part of the spec, and you may lose points if you ignore them. Such posts will always be placed in the Announcements section of the course page in addition to the Q&A page. (We promise that no official clarifications will be posted within 24 hours of the due date.)

What to turn in

Note: You will need to complete the "Academic Dishonesty policy questionnaire," found on the Assignments page on Canvas, before the submission link will be visible to you.

Please submit, on Canvas, the zip file that is created by the SpecChecker. The file will be named **`SUBMIT_THIS_hw3.zip`**. and it will be located in whatever directory you selected when you ran the SpecChecker. It should contain one directory, **`hw3`**, which in turn contains four files, **`ConnectGame.java`**, **`GameFileUtil.java`**, and **`Grid.java`**. Please LOOK at the file you upload and make sure it is the right one!

Submit the zip file to Canvas using the Assignment 3 submission link and **VERIFY** that your submission was successful.

> We recommend that you submit the zip file as created by the specchecker. If necessary for some reason, you can create a zip file yourself. The zip file must contain the directory **hw3**, which in turn should contain the three required files. You can accomplish this by zipping up the **src** directory of your project. **Do not zip up the entire project**. The file must be a zip file, so be sure you are using the Windows or Mac zip utility, and NOT a third-party installation of WinRAR, 7-zip, or Winzip.