This coursework is about maintaining and extending a re-implementation of a classic game called 2048. The new implementation has never been completed, but at least it runs, once it is set up properly. The game 2048 is played on a simple 4X4 grid with sliding numbered tiles that may be moved with the four arrow keys. Each turn, a new tile with a value of either 2 or 4 randomly appears in an unoccupied space on the board. The user's score starts at zero, and is increased whenever two tiles combine, by the value of the new tile. When a 2048-valued tile arrives on the board, the game is won. Players can keep playing after that to earn higher scores. The game is over when there are no more legal moves left for the player (i.e., there are no more empty spaces or adjacent tiles of the same value). More information about the original 2048 game and its history is available on Wikipedia (https://en.wikipedia.org/wiki/2048_(video_game)). In addition, you will find many opportunities on the internet to play the original game online.

Assessment: The marks will be split as follows:

- 10% for git use (show your screenshot of GitHub and history on your version control)
- 30% for refactoring
- 30% for additions
- 15% for documentation (Javadocs + class diagram)
- 15% for a video, explaining your refactoring activities and additions

Note: You are gently reminded that we are at liberty to use plagiarism detection software on your submission. Plagiarism will not be tolerated, and academic offences will be dealt with in accordance with university policy and as detailed in the student handbook. This means you may informally discuss the coursework with other students, but your submission must be your own work. Please also note that it is not permitted for you to copy and paste text from another source without correct referencing. If you are unclear about this process, please discuss with the module convenors during one of our lab sessions or at the end of a teaching session.

Requirement Specification

This coursework is about maintaining and extending existing code. So, for the maintenance part you have to use the existing code as a basis, and not write your own game from scratch.

Basic Requirements (to pass the assessment with 40%):

1. Set up a PRIVATE git repository at GitHub and use it actively for version control activities.
2. Do some basic maintenance of the delivered code base (this should include things like providing a meaningful Readme.md file and Javadocs, organising files in a meaningful

way into packages, breaking up large classes in a meaningful way to support the idea of single responsibility, improving encapsulation, etc.)

3. Extend the delivered code base by adding:
   3.1 A module-info.java file
   3.2 A START screen with a colour theme choice for the GAME screen and a button that allows going to the GAME screen.
   3.3 A SCORE pop-up, appearing at the end of game.

Additional Requirements do some of the following:

A1. Refactor the code by adding some design patterns to enhance maintainability.
A2. Create a permanent high score list (using a file to store scores and player names)
A3. Add interesting levels to the game (based either on your own ideas or on the original game)
A4. Add meaningful JUnit tests
A5. Use build files
A6. Come up with your own groundbreaking idea ... surprise us :)

Assignment Submission and Organization

This section describes which files need to be submitted for assignment and how they should be organised. Remember, there are a LARGE number of students in the class, thus the organisation of your submission is very important. You are required to create a root folder called **COMP2042surnameFirstname**, where "surname" is replaced with your surname, likewise for "firstname", The COMP2042surnameFirstname folder contains (and organises) digital copies of all of the files that compose the assignment.

Assignment Report: A **Readme.md** file (max. 500 words), documenting the work you conducted (highlighting the key changes you made for maintenance and extension, where you made them, and why you made them). WARNING: If you do not mention it here, do not blame us later if we miss it.
Your report contains the following information:

1. Your name and student number,
2. How to compile the code to produce the application,
3. Where your Javadoc documentation is stored (the path to the directory)
4. A list of features that are implemented and are working properly,
5. A list of features that are implemented and are not working properly,
6. A list of features, if any, that are not implemented with an explanation of why they were not implemented,
7. A list of new Java classes that you introduced for the assignment,
8. A list of Java classes that you modified from the given code base,

The report also informs the reader if any unexpected problems arose during the course of the assignment.Feel free to add any information which you feel is relevant.

Design Diagram: A file called **Design.pdf** contains a high-level class diagram that shows the structure of the final version of your game (considering only classes (excluding fields and methods, unless they are relevant for understanding design principles/patterns), interfaces, relationships, and multiplicity). If you use software to reverse engineer your class diagram, make sure the delivered diagram is correct and follows the above requirements.

Source Code Documentation: A copy of your generated Javadoc documentation is required. Javadoc produces a series of linked HTML web pages in order to facilitate the browsing of project implementations. The HTML web pages produced by Javadoc are placed in a folder called **javadoc** that resides in the **COMP2042surnameFirstname** folder described above. Recall that the Java output contains: (1) a brief description of each class, (2) a description of each method including input and return parameters, and (3) the original source code. The new Java classes you write use the following **author tag** convention: @author FirstName Surname. All modified Java classes from the previous code base use the following author tag convention:

@author FirstName Surname-modified.

In addition to reading your README file, we will look at the Javadocs to find out how you maintained and extended the game. If it is not obvious from there, we might miss it. Also, we have only a limited amount of time to look at each coursework submitted. So, please make sure to provide informative but concise Javadocs.

Project Implementation Files and Folders Description: A zip file containing your entire local project folder. it needs to be possible to import (or open) and run your project in either eclipse or IntelliJ. To avoid disappointment later, test your final version on a different computer. This should help to uncover hardcoded path dependencies, which was a major issue in previous years. Name your zip file **SurnameFirstName_IDE_JavaVersion.zip**, where IDE represents the name of the IDE you used, and Java Version represents the Java version you used.

Version Control: Set up a PRIVATE remote git repository at GitHub and add my account (kooitt) into your collaborator list. Your remote repository needs to be named **COMP2042_CW_UserName** (e.g., in my case it would be COMP2042_CW_kcztcc). Then follow the setup instructions provided to "Push an existing folder" (i.e. do an initial push to upload files from your local to your remote repository). Now you are ready for coding with version control.

video shows your software running and then (for the main part) explains your refactoring activities and additions. You also have to highlight the **TWO** achievements you are most proud of.

Folder and File Organisation

Thus, when completing the submission of the assignment, you have a directory structure in your **COMP2042surnameFirstname** folder that looks like this:

README.md

Design.pdf

javadoc

surnameFirstnameDemo.mp4

project/SurnameFirstName_JavaVersion.zip