Project 4: 2048

The deadline for this project is **Monday, 17th of June**, end of day, anywhere on earth (AoE)[1] for the first task of the project and **Monday, 24th of June**, end of day, anywhere on earth (AoE) for the remaining tasks. You can find more information about the project submission in appendix A.

1 Overview

In this project, you will implement the popular sliding tile puzzle game 2048 in Java[2]. In this game, tiles are placed on an $m \times n$ (usually 4×4) grid. By sliding the tiles into a specific direction, tiles with the same associated value may be merged into a tile of higher value. The objective of the game is to obtain a tile with value 2048 this way.

In section 2, we discuss the precise rules of the game. Section 3 documents the individual assignments, requirements and deadlines. Appendix A elaborates the grading of the project. The expected functionalities of the game simulator are documented in appendix B. Appendix C documents the provided command-line interface. Edge cases regarding the tile sliding are explained in appendix D. Finally, appendix E explains how to set up your VSCode.

Figure 1: Graphical User Interface

2 2048

Every square of the $m \times n$ grid of the puzzle may contain a tile. Every tile on the board is associated with a value, which is a positive integer. The game starts with two tiles at uniformly random positions. Both of these tiles have a value of 2 with probability 90% and a value of 4 with probability 10%.

In each turn, the player must choose a horizontal or vertical sliding direction towards which all tiles are moved, if possible. The resulting state of the game is determined by three subsequent phases, which cannot be interrupted by the player. In the first sliding phase, depending on whether the direction is horizontal or vertical, every row (horizontal) or column (vertical) of the grid will have all of its tiles slide towards the end of the row or column in the sliding direction, until the row or column is contiguous. In the subsequent merge phase, adjacent tiles are merged into a single tile if they have the same value. For each row or column, all squares are processed one after the other, starting with the square at the end of the row or column in the sliding direction. If the current square contains a tile and the next square to be processed (if existent) contains a tile with the same value, then the tile of the next square is removed and the value of the current square's tile is doubled, effectively merging both tiles in the direction of sliding. This process repeats iteratively until all squares are dealt with. Lastly, a second sliding phase occurs that closes all gaps that were introduced by merging. A subsequent merge phase will not occur. Appendix D further elaborates this procedure.

After every turn, a new tile will be spawned at a random unoccupied position of the grid. Like the initially spawned tiles at the start of the game, this tile has a value of 2 with a probability of 90%, and a value of 4 with a probability of 10%. The game is over if no sliding direction has any effect. This happens if the board is full, so that the sliding phase has no effect, and there are no adjacent tiles with the same value in any direction, which also makes merges impossible. In all other cases, one can either slide tiles, or merge adjacent tiles to free space.

[1] `https://en.wikipedia.org/wiki/Anywhere_on_Earth`
[2] You can also play it online, e.g. here: `https://play2048.co/`

3 Assignments

This project consists of two conceptually different tasks: writing tests and implementing the simulator. The implementation of the 2048 simulator may be pursued over the entire course of the project, *the testing part may however only be dealt with during the first week.* In this time span, you are encouraged to not only implement the project, but to write your own tests. These tests will help you check whether your game simulator matches the specification, as there will be no public tests aimed at checking your simulator's correctness provided for the second phase of the project.

Details regarding the first milestone will be discussed in section 3.1. Section 3.2 discusses the implementation of the game simulator, which contains the game logic. Lastly, the bonus assignment is discussed in section 3.3, where you implement an automatic computer player for 2048.

3.1 Writing Tests (9 Points) **Milestone α**

During the first week of the project, your task is to write tests against the specification of the simulator interface.

Appendix B documents the expected behavior of a correct game simulator implementation. To grade and give feedback on your tests, we will run your tests with both correct and faulty simulator implementations on our server. We expect that for every of our faulty implementations, you write a test that a correct implmentation would pass, but the faulty implementation fails.

Restrictions

The grading of your tests is subject to certain restrictions:

- Your tests are only required to instantiate the simulator with a 4 by 4 grid.
- Your tests do not have to verify whether random events actually occur randomly or determinstically.
- Your tests are not required to check whether the preconditions are asserted by the simulator.
- With respect to the `run` method, your tests must only assert that the game ends in a state where no move is possible.

These restrictions only apply for the grading of your own tests (milestone α). Our own tests may check your implementation (milestone ω) beyond these restrictions, in particular with respect to the behaviour in presence of invalid preconditions. Hence, it is reasonable to also write tests which check correctness of the implementation beyond these restrictions.

Technical Remarks All tests used to grade this task must be implemented in the Java package `ttfe.tests` (no sub-packages). It is irrelevant to this end whether you use multiple classes or not. If you need to implement helper methods or classes for your tests, these must also be implemented here. Your tests will only have access to the initially existent classes, interfaces and their methods, in particular those of `TTFEFactory`. You can create an instance of the interface to-be-tested via the method `TTFEFactory.createSimulator`.

Please make sure that your tests verify the behaviour of the simulator in such a way that an `AssertionError` (or a sub-class thereof) is thrown whenever the implementation is recognized as incorrect. Since the self-written tests are graded you **may not use any tests from your peers**.

– Milestone α –

Solve all previous assignments until **Monday, 17$^{\text{th}}$ of June**, end of day, anywhere on earth (AoE).

3.2 The Simulator (6 Points) **Milestone ω**

The simulator contains the implementation of the game's logic and controls the course of the game. You must implement the simulator yourself. At the start of the game, the board is initialized, i.e., two random tiles will be placed at random locations and the first turn starts. Every turn has the same protocol: The player chooses a (valid) direction in which the tiles are moved and a new tile is spawned afterwards. This happens until the board is completely filled with tiles and no two adjacent tiles with the same value exist. In this case, the game terminates with the message "Game Over". This notification also tells the player the final score and the number of turns played, as seen in fig. 3.

The rules regarding the movement and merging of the tiles are the same as in the web variant of the game[3]. The interesting cases are depicted in appendix D. The score only increases when tiles are merged and increases by the total value of all generated merge tiles.

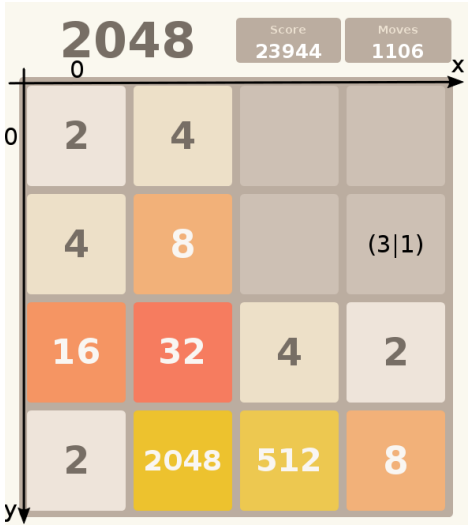


Figure 2: Coordniate System of the Simulator

Figure 3: "Game Over" Message at the End of the Game

Technical Remarks In the file `SimulatorInterface.java` you will find the documented simulator interface, which you shall implement with your own sub-class. The file also contains the specifications for the implementation and the tests in the form of Javadoc comments. Since you must write your own tests for those specifications, there are no publically available tests for these methods. It is therefore reasonable to implement the tests first (see section 3.1) before you proceed with the implementation of these methods. Please also note that all preconditions for these methods must be checked against: If a precondition of a method is not met when it is called, you must throw an exception as specified in appendix B.

You may create an instance of the simulator via the static method `createSimulator` which belongs to the class `TTFEFactory`. Implement this method in such a way that an appropriate instance of your own simulator interface implementation is returned.

Please pay attention that you only use the java package `ttfe` and sub-packages of this package for your implementation. Other packages will be disregarded by the testing infrastructure. Please also note that you neither modify nor extend the given interfaces, as they are used by the testing infrastructure.

Graphical User Interface (GUI) As soon as you implement your game simulation, you can play the game via the provided graphical user interface (see fig. 1). Calling the `main` function launches the GUI, expecting the inputs of a human player. Appendix C documents the commandline options you use to configure the program.

[3] `https://play2048.co/`

3.3 Bonus: The Computer Player (5 Points) Milestone ω

To finalize the project, you shall write your own comptuer player which attempts to "win" as many games as possible. Just like a human, the computer player may inspect the current state of the board and may also consider the future developments, depending on the complexity of your implementation. For a given state, your computer player must return a valid move.

While the board dimensions are not specified for the simulator, it is sufficient if your computer player is functional for a 4x4 board. Additionally, the game runs under a strict time limit: The game runs for at most 10 seconds[4], and is automatically terminated after this time has expired. The score accumulated so far is then the final score. How many project points you are awarded depends on how long your computer player survives on average. If your computer player successfully creates a 2048 tile in at least half of the tested games, you receive full points for this bouns task.

You may call arbitrary methods on the simulator object, in particular those that mutate the state of the board. However, before the selected move is executed, the state is reset to the previous state of the simulator. Note that the simulator may not place a piece at the same position as when you call the `addPiece` method, since tiles spawn at random positions, and calling this method yourself may advance the state of the random number generator.

Implement the computer player by implementing the player interface in `PlayerInterface.java` with your own sub-class. The computer player will subsequently be created via the method `createPlayer` inside the class `TTFEFactory`. Implement this method such that an instance of your own player is returned.

– Milestone ω –

Solve all previous assignments until **Monday, 24$^{\text{th}}$ of June**, end of day, anywhere on earth (AoE).

[4]The game is executed on our own hardware and without a user interface.

A Infrastructure, How to pass, Assorted Hints

Using dGit

Use git to clone your personal project repository from dGit, for example by entering

```
1 git clone ssh://git@dgit.cs.uni-saarland.de:2222/prog2/2024/students/project-4-{your matriculation number}
```

Submit your project using `git push` until Monday, 17th of June, end of day, anywhere on earth (AoE) for the first task of the project, and Monday, 24th of June, end of day, anywhere on earth (AoE) for the remaining tasks. If you need a refresher on how to use git, consult the materials from the Programming 2 Pre-Course or additional material online[5].

Grading

Similar to the way you obtain your project, you will submit the project through git. The last commit pushed to our server before the respective task deadline will be considered for grading of the task. Please ensure that the last commit actually represents the version of your project that you wish to hand in.

Once you `git push`ed your project to our server, we will automatically grade your implementation on our server and give you feedback about its correctness. You can view your latest test results by following the link to the Prog2 Leaderboard in dCMS Personal Status page. Milestone α uses two types of tests: **regular** tests and **eval** tests. Milestone ω uses **regular** and **eval** tests, as well as one additional **public** test. The public test is available to you from the beginning of the project in your project repository. We regularly run the regular tests on our server for your implementation, disclosing to you the names of the regular tests and whether or not you successfully pass them.

Eval tests are similar to regular tests, but run only once after the project deadline. They directly determine the number of points your implementation scored.

Notes

(a) Attempts at tampering with our grading system, trying to deny service to, purposefully crash, or extract grading related data from it will (if detected) result in your immediate expulsion from the lecture and a notification to the examination board. In the past, similar attempts have led to us filing criminal complaints with the police.

(b) There are no public tests provided for milestone α. You should use the tests you wrote for milestone α as well as the regular tests from milestone ω to test your simulator implementation.

(c) Regular test results for milestone ω become available as soon as milestone α has passed.

[5] Such as `https://git-scm.com/book/en/v2`

B Simulator Interface

The following table specifies the expected behaviour of the mentioned methods of the simulator. In the column Precondition (`Exception`) we state the necessary preconditions for the method to execute successfully. If the method is called with the precondition unmet, an instance of the specified exception class should be thrown.

For example, when calling `addPiece()` on a full board, an `IllegalStateException` should be thrown.

More documentation is to be found in the Javadoc comments inside the source code.

Method name	Specification	Precondition (`Exception`)
Constructor	Creates a new instance of the simulator, which represents the initial state of the game. Two tiles of valid value are already placed on the grid.	Board width and height of at least 2, random number generator id not equal to `null`. (`IllegalArgumentException`)
`addPiece`	Spawns the new tile at a random position on the board and chooses the value randomly as specified above.	The board is not full. (`IllegalStateException`)
`getBoardWidth`	Returns the width of the board as specified on construction.	-
`getBoardHeight`	Returns the height of the board as specified on construction.	-
`getNumMoves`	Returns the number moves made so far.	-
`getNumPieces`	Returns the number of tiles currently on the board.	-
`getPoints`	Returns the current score.	-
`getPieceAt`	Returns the value of the tile at the given coordinate (see fig. 2), or 0 if there is no tile at this position.	Coordinates must be within the board dimensions. (`IllegalArgumentException`)
`setPieceAt`	Places a tile of specified value at the given coordinate. If there is a tile already of this position, it will be overridden. If a value of 0 is specified, the tile at this position is removed instead, if existent.	Coordinates must be within the board dimensions. Value must be non-negative. (`IllegalArgumentException`)
`isMovePossible`	Returns `true` if and only if there is a valid move direction, or if and only if the specified move direction is valid.	The specified move direction is not `null`. (`IllegalArgumentException`)
`isSpaceLeft`	Returns `true` if and only if there is at least one unoccupied space.	-
`performMove`	Attempts to execute a move in the given direction and return `true` if and only if this was successful. Increases the current score if needed.	The specified direction is not `null`. (`IllegalArgumentException`)
`run`	Starts the game and communicates its progression via the `UserInterface`. In every state of the game, the move in this state as specified by the player must be executed. The successor state must be consistent with that move. In the final state, no moves may be possible.	Player and user interface are not `null`. (`IllegalArgumentException`)

C Command-Line Interface

As soon as you implemented the simulator, you may play the game yourself via the graphical user interface. To this end, run the main method of the class `TTFE.java`, for instance using the provided VSCode configuration shipped with your project.

A command-line interface has already been implemented and is documented below. Four optional parameters can be passed to the program in arbitrary order:

(a) `--seed N` Sets the seed for the random number generator (Default: `N` = 4711)

(b) `--width W` Sets the width of the board (Default: `W` = 4)

(c) `--height H` Sets the height of the board (Default: `H` = 4)

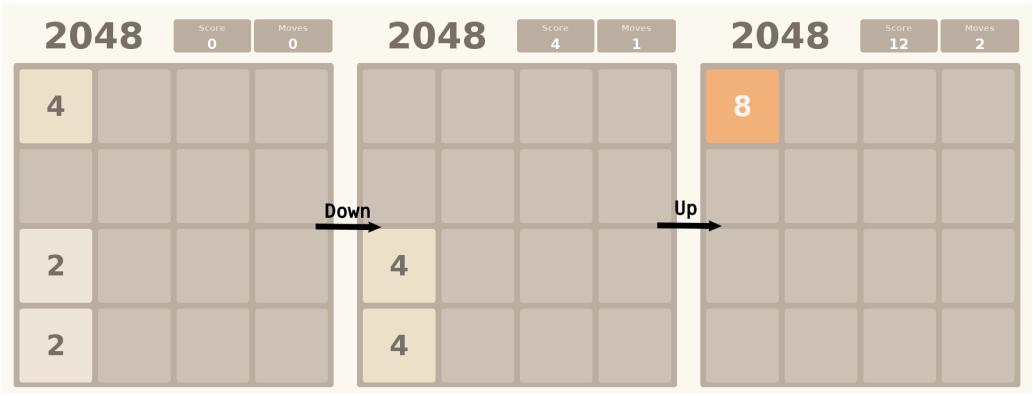
(d) `--player [h|c]` Set whether a `h`uman or `c`omputer is playing (Default: `h`)

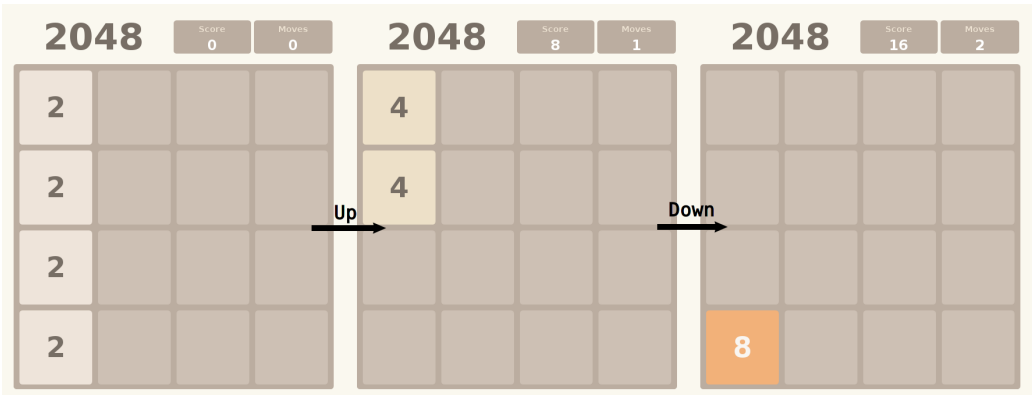
When playing the game using the provided graphical user interface, you can control the game with the arrow keys, just like the web version.

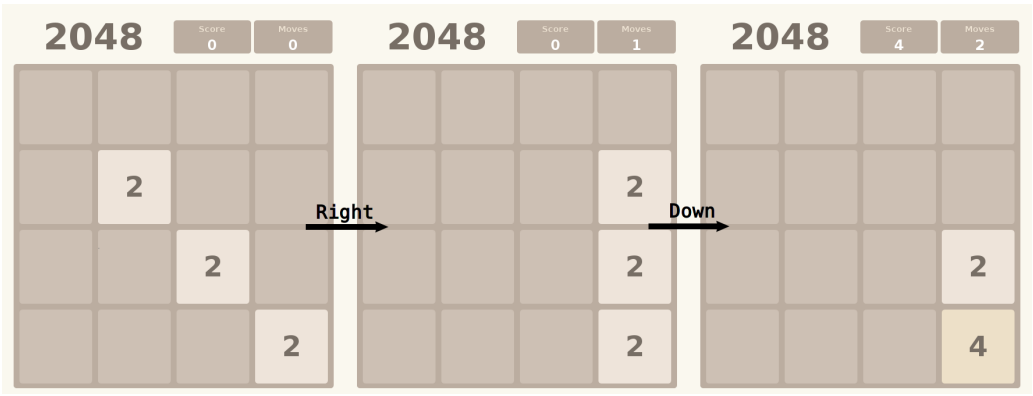
D Simulator Edge Cases

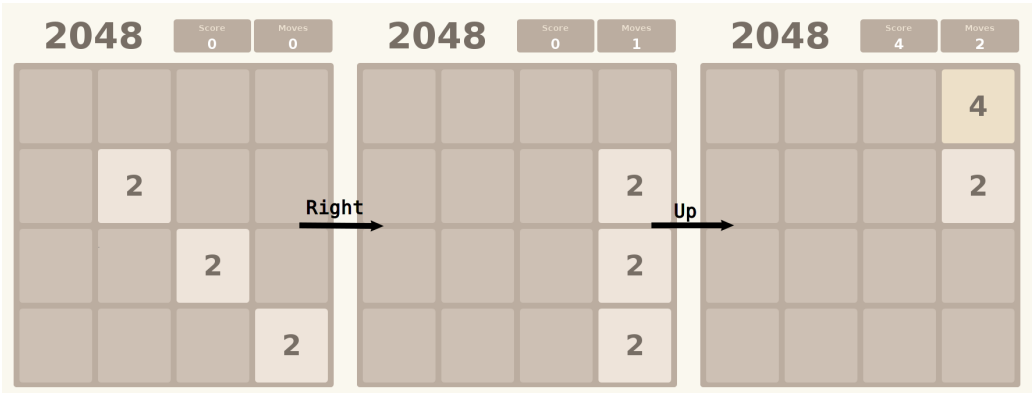
The following examples document the edge case behavior your simulator should implement.

Note: We omitted newly spawned tiles for the sake of simplicity.









E Compiling and Testing

Starting with this project, you will implement the following projects in Java. We recommend that you use Visual Studio Code to work on the projects, as well as installing the extension *Extension Pack for Java*, which contains many useful features for working with Java in VSCode.

In order to run Java applications on your machine, you need to install a Java Development Kit (JDK). The JDK used on our servers is the Temurin 21 JDK (LTS). We recommend installing the same version, so you can reproduce the compilation results of our server. Our installation guide[6] documents how to install Temurin 21 JDK (LTS). Using the regular tests, you can convince yourself that your implementation behaves on our servers as it does for you on your local machine.

Compiling in Visual Studio Code

Visual Studio Code automatically compiles the projects when you run tests or start with a main method. In contrast to the C projects, you therefore do not have to execute a dedicated build command.

Testing in Visual Studio Code

The testing framework we are using is called JUnit, which VSCode should install automatically. You can identify whether JUnit is already installed on your system by observing whether the `org.junit.*` imports in the file `src/ttfe/publictests/TTFEFactoryTests.java` get correctly resolved, and whether a green tick mark (or red cross) appears next to the class and `@Test` declarations, see fig. 5.

In the case that JUnit is not already installed on your system, you need to manually add `JUnit` and its dependency `hamcrest` as dependencies to your project.

Adding JUnit Manually to Your Project

1) Download `junit-4.13.2.jar` and `hamcrest-core-1.3.jar` from the materials page of the dCMS.
2) Navigate to the Java Projects side bar, scroll down to "Referenced Libraries" and press "+" to add Jar libraries to project classpath, as shown in fig. 4.

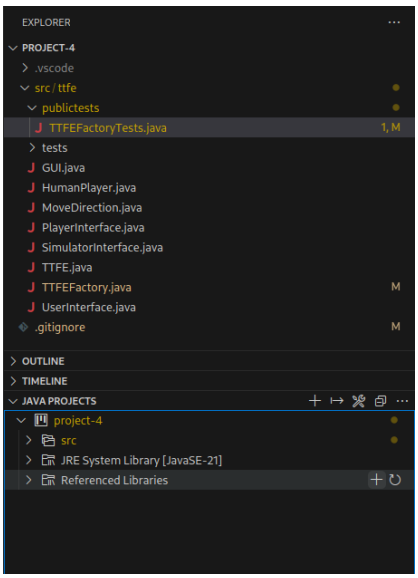


Figure 4: Adding Libraries to Project

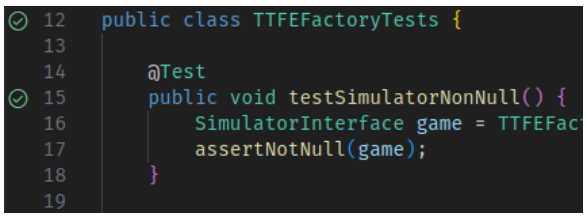


Figure 5: Successful JUnit Integration

3) Select the two `.jar` files and confirm the dialog.
4) Click on "Referenced Libraries" to check that the libraries have been added and restart VSCode.
5) You can verify that you successfully added JUnit to your project if the green tick marks (or red crosses) depicted in fig. 5 appear.

Clicking the tick (cross) icon next to the `public class` declaration lets you run all tests annotated with `@Test` inside that class. Note that, before you complete your simulator implementation, your tests will fail locally, as the functions under test are not implemented yet.

[6] `https://dpad.cs.uni-saarland.de/aSh5cX5vQZaa0s1IyyV7hg`