2048

Vicki D. Bealman
Full Sail University

Software Requirements Specification Document

Version: (1) **Date: (12/08/2019)**

Table of Contents

1. Introduction

2048 is a simple, open-ended, addictive puzzle game which involves shifting tiles around a grid in an attempt to merge simlar tiles to score points. The game especially resonates with computer science folk as the scoring system is based on powers of 2. You will build the underlying classes to implement a slightly expanded version of 2048 in this assignment and analyze the complexity of some of the operations required for the game.

1.1 Purpose

This document specifies the requirements for the 2048 game software. These requirements relate to the functionality, constraints, performance, attribute and the system interface.

The 2048 is a program used to play the 2048 game. First goal is to allow two users or players to play the game. And the second goal will be that the program should be working and allow the users to play the game.

1.2 Scope

2048 is a simple, open-ended, addictive puzzle game which involves shifting tiles around a grid in an attempt to merge simlar tiles to score points. The game especially resonates with computer science folk as the scoring system is based on powers of 2. You will build the underlying classes to implement a slightly expanded version of 2048 in this assignment and analyze the complexity of some of the operations required for the game.

1.3 Definitions, Acronyms, and Abbreviations.

`Tile`

Abstract class which has the concrete implementation `TwoNTile`. This class tracks a game piece which is moved around a board. If a moving tile collides with another tile, they may merge. In the case of `TwoNTile`, merging happens if two colliding tiles have the same value: two 2-tiles would merge, two 4-tiles would merge, but a 2-tile and a 4-tile would not merge. `Tiles`provide methods to determine if they merge and to produce

`Board`

Abstract class with the concrete implementation `DenseBoard`. This class is responsible for keeping track of a grid of tiles of arbitrary rectangular dimensions. The main operation of the Board is to perform a shift to the left, right, up, or down which moves tiles in the specified direction as far as they will go. If two tiles collide they may merge depending on their properties. The board allows tiles to be added and retrieved according to certain semantics that enable random tiles to be placed.

The concrete implementation of `DenseBoard` will keep track of tiles using a 2D array or `ArrayList`. Since 2048 boards never grow or shrink, there is not much difference between using standard arrays or `ArrayList` in this case so long as there is one memory slot available for every possible tile position.

`Game2048`

Concrete class. Provides a high level interface to a complete game of 2048 with methods to manipulate a board, report the score, and detect the Game Over Conditions. User Inteface classes will interact with this class. A number of the methods of `Game2048` simply call corresponding methods of the game's internal `Board`.

`PlayText2048` and `PlayGUI2048`

Implement a simple text-based and GUI user interface to play 2048 through its`main()` function.

1.4 Intended Audience and Document Overview

This SRS document is intended for developers, testers, and end users. It contains an overall description of the software, followed by specific requirementsand also non-funtional ones. It is recommended for the reader to begin with the overview sections and proceed with the sections most pertinent.

1.5 Document Conventions

This document follo0ws the IEEE formatting requirements, Times New Roman 12-point font was used in the majority. .

Overall Description

2.1 Product Perspective

2048 is a single-player sliding block puzzle game. The game's objective is to slide numbered tiles on a grid to combine them to create a tile with the number 2048. However, one can continue to play the game after reaching the goal, creating tiles with larger numbers.

2048 is written as free and open-source software subject to the MIT license, written in C#.

2.1.1 Product Funtionality

A summary of the primary functions 2048 offers is as follows:

- Generate a new 2048 grid.
- Print the current grid.
- Print 4 grids.
- Open and resume a previously saved grid.
- Save the current grid.
- Download new versions of the game.
- Change language.
- Enter numbers on the grid.

- Clear previously entered numbers and tiles.
- Hide doubled tiles.
- Clear previous moves.
- Automatically resolve the current grid.
- Exhibit current score.
- Exhibit high score.
- Restart button.

2.1.2 Operating Environment

2048 runs on Android, iOS, and Windows platforms, in platforms with a Java Runtime Environment, Java version greater or equal to 1.6 is required.

2.1.3 User Documentation

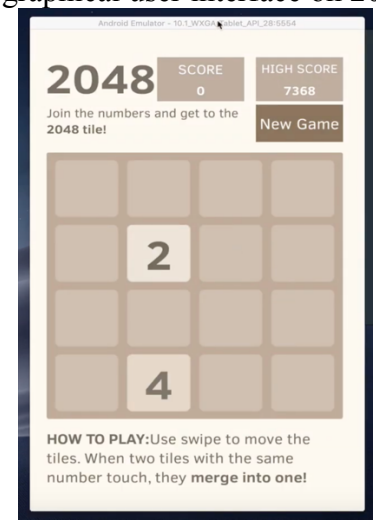
The following documents are available in text format:

- COPYING: a document listing public license, copyright, terms, and conditions of use.
- HOW-TO-RUN-2048: informs the end user how to run 2048 on his/her system
- NEWS: offers a link to the project's forums and explains different versions of the development status.
- README: offers the project's changelog and lists various websites referencing it.

Specific Requirements

3.1 User Interfaces

- This is the primary graphical user interface on 2048:

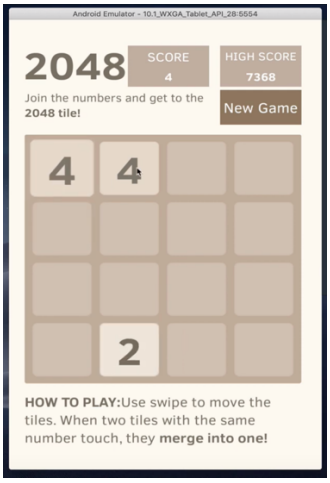


The current grid occupies a majority of the window, and on the top we have:

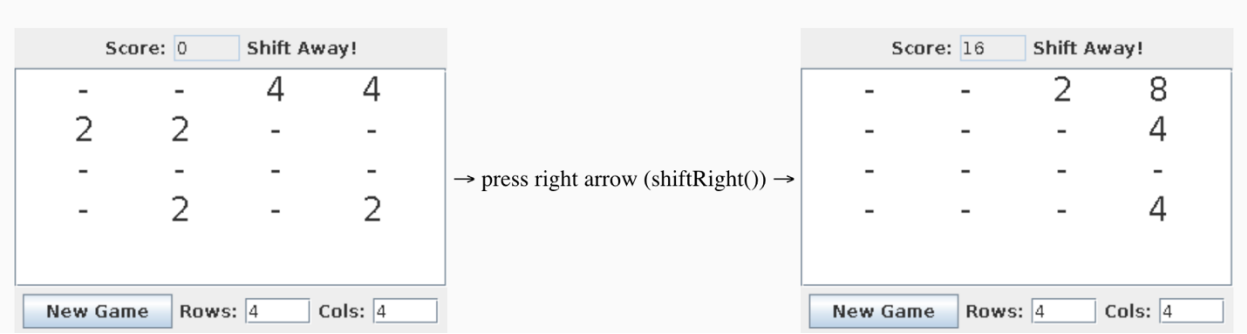
- Title,
- Description,
- Score, and
- High Score.

On the bottom is the "How To Play" instructions.

Swiping and moving two tiles with the same number merges tiles into one, adding numbers together.



PlayGUI2048: A graphical interface is provided which relies on Game2048 to function. This class does not require modification and should work after it allows the arrow keys to be used to speed play. The board size can be set on starting a new game by entering appropriate information.



3.2 Functions

Functional requirements define the fundamental actions that must take place in the software in accepting and processing the inputs and in processing and generating the outputs.

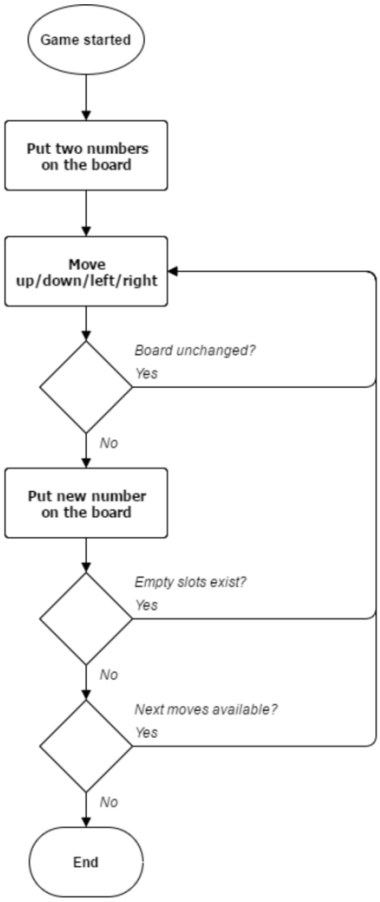
These include:

- *TwoNTile:*
    - Concrete implementation of a Tile. TwoNTiles merge with each other but only if they have the same value.
- *DenseBoard*:
    - Tracks the positions of an arbitrary 2D grid of Tiles. DenseBoard uses an internal, multi-dimensional array to store the tiles and thus has a O(R * C) memory footprint (rows by columns).
- *Game2048*
    - Represents the internal state of a game of 2048 and allows various operations of game moves as methods. Uses TwoNTiles and DenseBoard to implement the game.
- *PlayText2048*:
    - Class to play a single game of 2048.
- *Shifts without Merges*
    - When shifting tiles in a particular direction, all tiles on the board that can move do so in the direction indicated. The following examples do contain any merges of tiles.
- *Shifts with Merges*:
    - During a shift, tiles are thought to be "moving" in the shift direction and colliding with one another. For TwoNTiles, collisions between two tiles with the same value causes them to merge and creates a new tile with a larger value. Merging happens even when tiles haven't actually "moved" through any empty space but moving in a direction would cause a collision with another tile of the same value.
- *Resolving Merges*:
    - Merges are resolved in the opposite direction of the shift direction.
        - Shifting left resolves merges from left to right
        - Shifting right resolves merges from right to left
        - Shifting up resolves merges from up to down
        - Shifting down resolves merges from down to up
- *Shifts that Do and Do Not Change the Board*:
    - Some shifts move tiles while others do not. In the 2048 game mechanics, performing a shift that moves tiles results in another tile being added to a random free location on the board. Shifts that do not not move any tiles do not move the game forward: no tiles are added, no score is gained, and the state of the board remains the same.
    - To facilitate this in the game, Boards must indicate when the last shift has moved tiles. During a shiftLeft(), shiftRight(), etc., the Board should track if any tiles have actually been moved. Merging tiles counts as moving. After the shift operation completes, a subsequent call to Board's lastShiftMovedTiles() method should return true if any tiles moved and false otherwise.
    - A freshly created Board returns false if no shifts have been performed.
- *Scoring of Shifts*:

- Players score points every time a shift merges tiles. The score for merging two TwoNTiles is the sum of their values which also happens to be the value of the tile they produce.
    - Merging two 2-tiles gives a score of 4 and produces a 4-tile
    - Merging two 4-tiles gives a score of 8 and produces an 8-tile
    - Merging two 8-tiles gives a score of 16 and produces an 16-tile
    - The return value of the DenseBoard shift methods (shiftLeft(),shiftRight(), etc.) is the score for shifting which is the sum of all merges that happen.

3.3 Game Loop

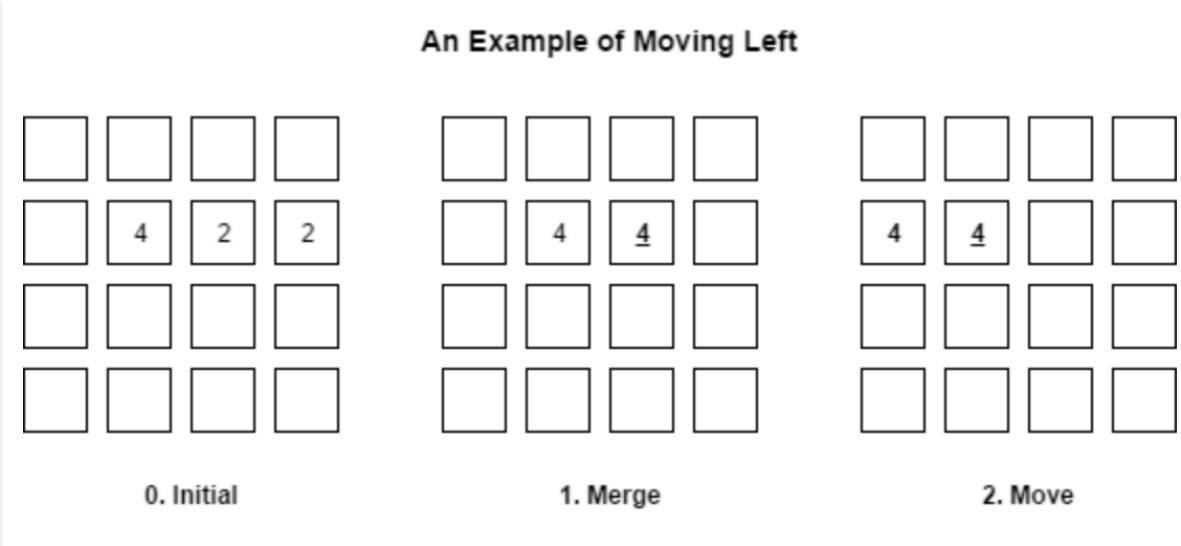
Everything starts from the game loop. We have 2 numbers on the board whose positions are randomized. When numbers are moved towards one of 4 directions, a new number will be added, which can be 2 or 4. In the mean time, a pair of the same number on the same row or column will be merged and summed up. If the board is full and no more moves are available, the game ends.



3.4 The Four Moves

Moving the numbers on the same row or column can be broken down into two phases:

1. Merge and sum up every pair of same nearby numbers, or same numbers that are not interrupted by other numbers.
2. Move remaining numbers toward the direction.



- *MoveLeft* is the most simple one to implement. We have a method that operates a row where phase one is scanning and summing up every pair of same numbers, and phase two is moving the remaining numbers to the left.
- MoveRight is very similar to MoveLeft where two for-loops operate oppositely.
- *MoveUp* and *MoveDown* is either vertical or horizontal.

3.5 Game Over Check

There are various ways to check whether the game has reached the end because of no more valid moves.

- In this implementation, we adopt the most simple way in which we copy the original board state to another instance and check if the four moves can modify the new array.

3.6 Performance Requirements

This subsection specifies both the static and the dynamic numerical requirements placed on the software or on human interaction with the software, as a whole. Static numerical requirements may include:

- One terminal to be supported
- One user to be supported
- The AI in its default configuration (max search depth of 8) takes anywhere from 10ms to 200ms to execute a move, depending on the complexity of the board position. In testing, the AI achieves an average move rate of 5-10

moves per second over the course of an entire game. If the search depth is limited to 6 moves, the AI can easily execute 20+ moves per second.

3.7 Logical Database Requirements

This section specifies the logical requirements for any information that is to be placed into a database. This may include:

- Swipe user input.
- Used every play user makes.
- Acessible by swipe or L-R-U-D arrows

3.8 Design Constraints

3.8.1 Standards Compliance

- No existing federal or regulatory constraints.

3.9 Software System Attributes

3.9.1 Availability

- System runs only infrequently on-demand.
- The system shall allow users to restart the application after failure with the loss of available empty tiles.

3.9.2 Security

- No cryptographic techniques
- No restricted communications between areas of the program

3.9.3 Portability

Attributes of software that relate to the ease of porting the software to other host machines and/or operating systems. This may include:

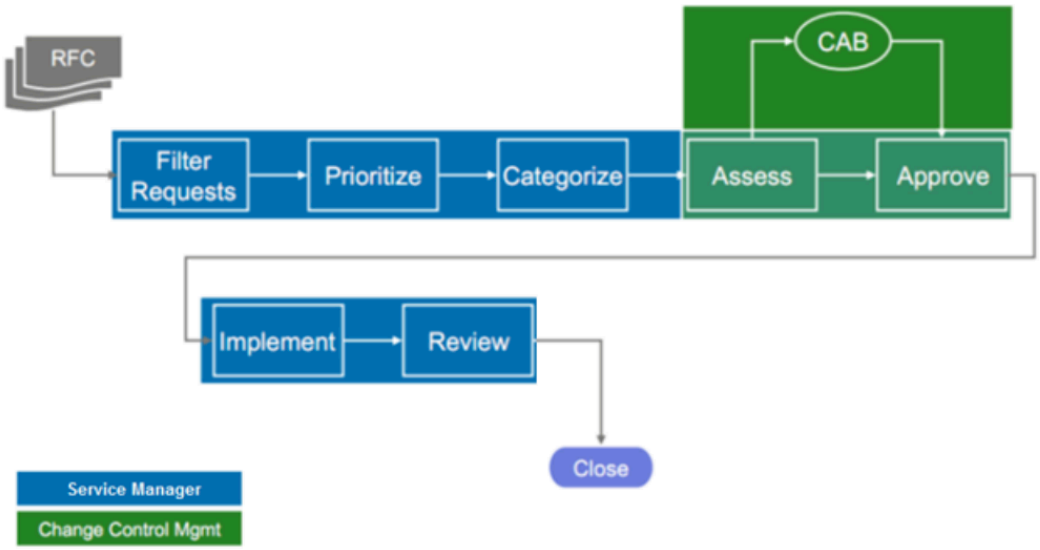
- 0% of components with host-dependent code
- 0% of code that is host dependent
- Java is the proven portable language
- Android and iOS operating system

Change Management Process

Change Management includes the following processes:

1. Change Logging
2. Change Review
3. Change Assessment and Planning
4. Change Approval
5. Coordinate Change Implementation
6. Change Evaluation and Closure
7. Emergency Change Handling

The following graphic depicts a simplified Change Management flow:



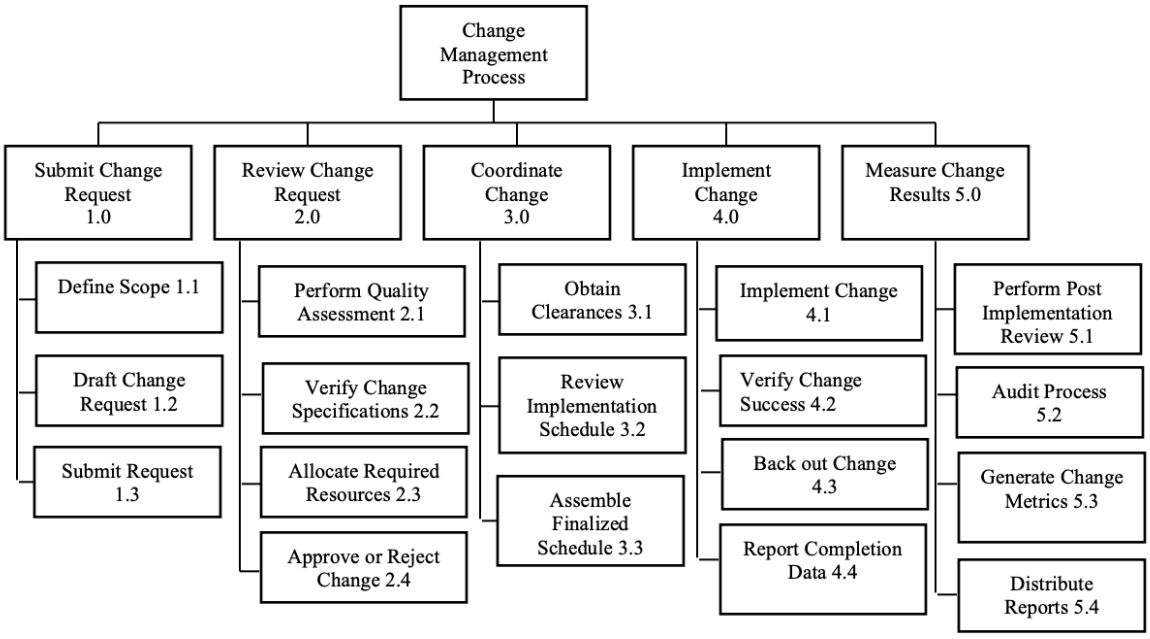
The emergency change process differs from a normal change process in:

- Approval is given by the authorized Emergency Change Approvers rather than waiting for a regular CAB meeting
- Testing may be reduced or in extreme cases eliminated, if necessary to deliver change immediately
- Updating of the change request and configuration data may be deferred, until normal business hours

Emergency Changes require:

- Technical Change Approver review and approval
- Submission of a Change Request within one business day after the is

4.1 Change Management Process



4.2 Change Categories

There are three main designated categories of Change Requests (Normal, Filtered, and Emergency). These categories are based on both the Risk Level Assessment (RLA) and time between submission of a Change Request, and the Start Date of the change. Change Categories are reviewed in PIR and reported in Change Management metrics.

- *Normal Changes* are defined as changes that meet required lead-time, submission cut-off time, and maintenance window (if applicable). Normal Changes must follow all Change Management Procedure activities, unless they are defined and approved as Filtered Changes.
- *Filtered Changes* are a pre-defined subset of Normal changes that have been identified as having no impact, or outside the scope of the Change Management process. Filtered Changes require the submission of a Filtered Change Request for tracking and recordkeeping purposes but will not have an associated approval process.
- *Emergency Changes* are defined as changes required to immediately restore service or to avoid an outage where no other workaround is available. Upon approval, a Change Request may be entered after change implementation.