Requirements and Analysis Document for

PaintIT

Henrik Lagergren, Markus Pettersson, Aron Sjöberg,
Ellen Widerstrand, Robert Zetterlund

2/10/18

v.1.0

Contents

1 Introduction

PaintIT is an interactive game for two players. Where one player is the painter and is given a canvas and a word to depict. The other player is the guesser and is shown the the finished canvas from the painter and then guesses which word is depicted.

1.1 Definitions, acronyms, and abbreviations

1.1.1 General words used throughout document and development

- MVC - Model View Controller.
- JavaFX - The standard GUI library for Java.
- UI - User Interface.
- Design Pattern -

1.1.2 Words explaining the game

- **Painter** The player that is being presented a word and that is supposed to paint on the Canvas.
- **Word** The word that is supposed to be painted and guessed.
- **Canvas** The canvas contains the painting of the painter, the canvas is also shown to the guesser.
- **Guesser** The player that is being presented with a painting and eight tiles.
- **Tiles** A tile consists of 1 (one) letter, in total there are 8 (eight) tiles that has n amount of letters being of the words, (where n is and int of the amount of letters in a word)
- **Round** - A round consists of a word being presented, painted and guessed, either incorrectly or correctly.
- **Streak** - The amount of successful rounds, meaning the painting was guessed correctly.

1.1.3 Implementation definitions

- **TopController** - Is a controller that shows and prepares views, unlike every other controller, it does not hold one (1) view, it holds all the views in the application within a list.

- **Canvas** - Not an actual class. The canvas is the umbrella term for the area which you can paint on. The Canvas consists of the CanvasModel, CanvasController and the CanvasView.
  - **CanvasModel** - The actual representation of the canvas. The CanvasModel consists of a 2D matrix of type Color. The model is observed by the CanvasView.
  - **CanvasController** - The CanvasController recieves coordinates from the user via the CanvasView, and changes the model accordingly with regards to the equipped Tool.
  - **CanvasView** - The actual view of the canvas, it extends the JavaFX Class Canvas and uses a pixelWriter to change pixels.
- **Tool** - [REDACTED] Currently an interface which is implemented by Brush, SprayCan, Eraser. Tools is observed by CanvasController and calls the update(int x, int y, Color color).
  - **Brush** -
  - **SprayCan** -
  - **Eraser** -

2 System architecture

The game consists of a startup stage and a drawing - guessing loop. The game is visualised to the user with GameScreens which are changed throughout the game.

2.1 Overview

2.1.1 Game Startup

2.1.2 Drawing and Guessing

2.1.3 The Score

2.1.4 Game Dictionary

2.2

2.3 Subsystem decomposition

Describe in this section each identified system component (that you have implemented).

2.3.1 StartScreen Component

The StartScreen component consist of several views. The views are fairly simple and therefore do not need any models since they only direct the user to different parts of the application. To be able to change views whenever a button is pressed every view has an instance of GameSession, which in turn tells the TopController to change the current view.
Different design patterns has been implemented to improve the communication between application and user. The mainMenuview, GameSetUpView and WordRevealView has a ”prominent done button” to put emphasis on the primary or most important action. SettingsView has the settings-pattern, which gives the user a central place to specify preferences for how the application should behave.
When starting the program, the user is faced with the Main Menu, a simple view with a clean design. The user can get information on how to play the game, view the high score and start a new game, through three different buttons.

2.3.2 Settings Component

2.3.3 Canvas Component

The Canvas Component component consists of the classes CanvasModel, CanvasView, CanvasController. Essentially the component responsibility is handling the painting-aspect of the application.

The CanvasComponent uses an MVC structure, effectively resulting in the need of an Observer and Observable interface which is implemented in CanvasView and CanvasModel respectively. See figure below (). Using the javafx library results in actionevents being handled in the CanvasView, the implementation is as follows:

1. User clicks somewhere on Canvasview
2. The appropriate actionevent is solved using the javaFX library and calls appropriate method in CanvasController.
3. The CanvasController updates the model accordingly, using the equipped tool.
4. The model is updated, and calls on update on the CanvasView using the Observer Pattern.

2.3.4 GuessingView Component

The GuessingView is a JavaFX view that instantiates two JavaFX components, CanvasView and TileBoardView.
The TileBoardView visualizes the guess-process of the game. The data that it visualizes is the Tile[] guessWord and Tile[] availableTiles from the GuessLogic Object. These Arrays are used to Give data to two arrays of TileSlots. One array containing the tiles in the guess and one containing the available tiles. The TileSlot is a small AnchorPane with a button inside. This is the tile that the user sees.

Whenever one of the tiles is clicked it modifies the Guesslogic (model) through the TileBoardController where the EventHandlers for the buttons are.

GuessLogic is the Class where the logic for the Guessing part of the game is Stored. It has methods that add or remove Tiles from the Guess as well as checking when a guess is correct. GuessLogic implements an Observer Pattern and whenever the data in GuessLogic is modified it notifies its list of observers that a change has been made. TileBoardView is one of the observers and it reads data from the backend and updates whenever notified by the Observarable GuessLogic.

2.3.5 ScoreBoard Component

What is this component responsible for and what does it do. Divide the component into subsystems (packages) and describe their responsibilities. Draw an UML package diagram for the top level. Describe the interface and dependencies between the packages. Try to identify abstraction layers. Think about concurrency issues. 1 If your application is a standalone then:
Describe how MVC is implemented
Describe your design model (which should be in one package and build on the domain model)
- Give a class diagram for the design model. otherwise: MVC and domain model described at System Architecture Diagrams
Dependencies (STAN or similar)
UML sequence diagrams for flow.
Quality
List of tests (or description where to find the test)
Quality tool reports, like PMD (known issues listed here)
NOTE: Each Java, XML, etc. file should have a header comment: Author, responsibility, used by ..., uses ..., etc.

2.4 Game Component

As above, and continue for all components.

2.5 Keyboard Component

2.6 DoneView component

The DoneView component compiles data from both the current round of gameplay aswell as the ongoing game session. It is here were both players (the team) are presented with statistics of how well they are doing and they also get faced with a descision wether to keep playing or not.

The data presented in this view is accessed through an instance of GameSession, which is passed as an argument to the constructor of DoneView. This is the only reference DoneView holds to any other object. It does however depend usage of certain classes and interfaces outside of it's own package, such as the abstract class ButtonFactory and the interface GameScreen.

The DoneView is composed of x labels and y buttons, and does not contain any other view within itself (as seen with PaintingView, GuessingView ..). The DoneView does not modify any data, only accessing data that is being passed-by-value. Since there are no other views active at the same time as DoneView there should not be any problems regarding concurreny, e.g. dead locks or race conditions. All data DoneView presents is fetched during the call of method apply();, which GameSession is responsible of calling before DoneView is shown.

How the DoneView component singjalz to GameSession when it is done is like with all other GameScreens. DoneView has one button with a String stored inside of it, which it will pass to the show(String URl); method in GameSession. This button is dedicated to taking the next, painting player back to the WordRevealView, where he/she will be presented with a new word and a new round of gameplay starts.

3 Persistent data management

How does the application store data (handle resources, icons, images, audio, ...). When? How? URLs, pathes, ... data formats... naming..

4 Access control and security

Different roles using the application (admin, user, ...)? How is this handled?

5 References