System design document for
Trivial Pursuit - Around the World

Version: 2.1

Date: 2015-05-31

Author: Emina Hromic, Ina Tran, Heléne Jarl and Rasti Tengman

This version overrides all previous versions.

1 Introduction

1.1 Design goals

1.2 Definitions, acronyms, and abbreviations

2 System design

2.1 Overview

2.1.1 Afterburner.fx and view, how it works

2.1.2 What happens when the application starts

2.1.3 How events are handled

2.2 Software decomposition

2.2.1 General

2.2.2 Layering

2.2.3 Dependency analysis

2.3 Concurrency issues

2.4 Persistent data management

2.5 Access control and security

2.6 Boundary conditions

3 References

APPENDIX

1 Introduction

1.1 Design goals

The aim is to have as much loosely coupled classes as possible, this by implementing the Model-View-Controller design pattern to the highest extent. The goal is also to have as understandable code as possible so that no comments are needed. It should also be easily tested without having to create any unnecessary dependencies.

1.2 Definitions, acronyms, and abbreviations

- **MVC** - The design pattern Model-View-Controller. This is used for splitting the application in such a way that part's purpose is to handle its area of work.
- **GUI** - graphical user interface.
- **TP** - Trivial Pursuit, the actual application.

2 System design

2.1 Overview

The Trivial Pursuit application is implemented as a JavaFX application and Afterburner.fx is used to handle all the classes. It is a complementary framework to JavaFX that supports dependency injection. With this the application is able to be implemented with MVC in a flexible way since Afterburner.fx supports this design pattern quite well. Even though MVC is well supported by this framework, the framework has some bugs that made it hard to implement the start view of some view classes. However, the solution to this was to use Initializable in the controller classes.

2.1.1 Afterburner.fx and View

As previously mentioned, Afterburner.fx is used in the application and here follows a brief description of how it works with views. Every implemented view class extends a class called BaseView, which is used as a base to set some standard settings for views in our stage. Below follows a couple of examples on what BaseView does to simplify things:

1. Only one stage is used throughout the whole application and BaseView e.g. sets the title of the stage and so that the stage size cannot be resized.
2. BaseView changes the content in the stage.

BaseView extends FXMLView which makes it easy to load the .fxml views, since it connects the view class to the right .fxml file. Certainly the files need to be named correctly.

2.1.2 What happens when the application starts

Since Trivial Pursuit is an JavaFX application, the main class (TrivialPursuit) needs to extend Application. When Application is extended a start method called *initialize* has to be implemented, which is runned when the application is started. In the main class the view to StartView is created, and from here the content of the stage is switched depending on which button the user presses. Thanks to Afterburner.fx, the content of the stage is able to be switched smoothly since it provides the method *injectmembers* which can be used for this.

Mainly throughout the application a view is not created until it needs to be used. To later on change the view after it already has been created, the method *show* is used.

2.1.3 How events are handled

The .fxml file is where everything that the user is able to see is built, such as all the buttons, labels, colors etc. Every .fxml file is connected to a certain controller. In the .fxml file, there are components that are supposed to generate an event. These are simply connected to a certain method in the controller class that the .fxml is connected to.

2.2 Software decomposition

2.2.1 General

The application is divided into four different packages:

- **trivialpursuit** - contains the main class which starts the program
- **model** - the core of the program that possesses most of the logic.
- **view** - contains all the graphics in the program
- **controller** - handles all the events and modifies model

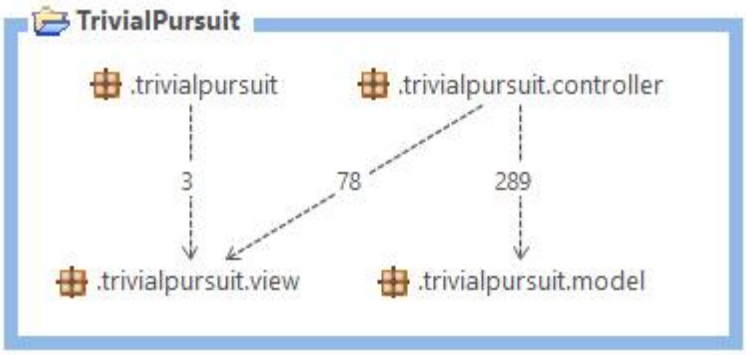

Image 1: the external package diagram

2.2.2 Layering

The layering can be seen in Image 1 above.

2.2.3 Dependency analysis

There are no circular dependencies. See Image 1 for the dependency analysis.

2.3 Concurrency issues

N/A. TP is a single thread application.

2.4 Persistent data management

N/A. No data is saved during each session, which means that neither the games nor the scores of the game will be saved and can therefore not be used for future purpose.

2.5 Access control and security

The same access is applied for all of the users of this application. There are no restrictions or encrypted security.

2.6 Boundary conditions

There are no certain boundaries for the application. It is simply launched on the desktop by cloning the Git-repository and then run the main class, TrivialPursuit.

3 References

Trivial Pursuit rules: http://sv.wikipedia.org/wiki/Trivial_Pursuit

MVC: http://sv.wikipedia.org/wiki/Model-View-Controller

APPENDIX

UML class diagram for each package:

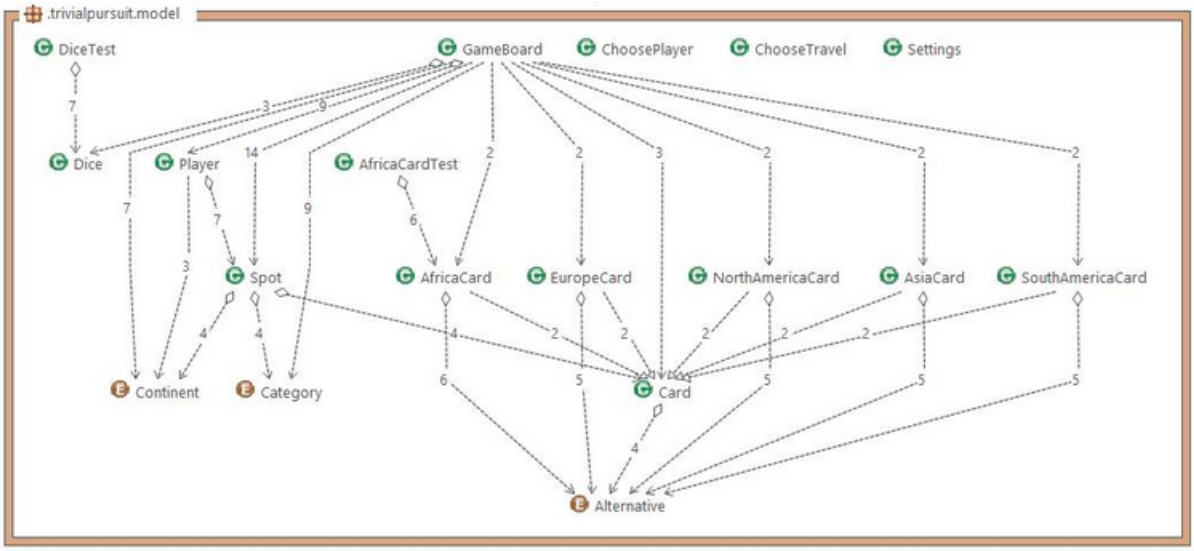

Image 2: the model package

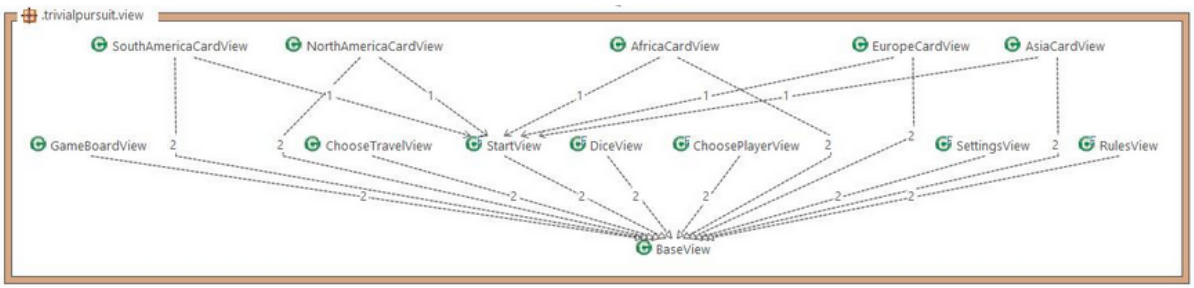

Image 3: the view package

Image 4: a part of the controller package

The reason for why the controllers do not have any relations to each other (see *Image 4*), is because they all are connected through .fxml files.