Virtual Arcade

System Design Document (SDD)

Version: 2.0
Date: 29/09 2020
Authors: *Georges Kayembe, Mhd Jamal Basal, Jonathan Stigson, Oliver Ljung, Robert Sahlqvist*

1 Introduction

1.1 Purpose of application

The project aims to implement a virtual arcade with five or more integrated retro games. The application has great focus on recreating a nostalgic feeling whilst introducing a modern vibe into old retro arcade games.

1.2. General characteristics of application

The program is a platform independent desktop application able to launch five different retro games. The five games available are as follows: Breakout, Frogger, Pong, Snake and Space Invaders.

1.3 Definitions, acronyms, and abbreviations

GUI: “Graphical User Interface” also “User Interface”, the part of the program that the user sees and interacts with.

DoD: “Definition of Data”, a list of requirements that have to met for a User Story to be considered done.

User Story: short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system.

RAD: “Requirement and Analysis document”.

SDD: “System Design Document”.

UML: “Unified Modeling Language”, A visual language for mapping and representing systems. Can be used for modelling och class diagrams, sequence diagrams and more. In this project used for domain and design models.

MVC: ”Model, View, Controller”. A well known design pattern which is hugely used in applications that have a GUI representation. It aims to separate the different areas of code(logic, viewing,...)

FXML: An XML-based user interface markup language created by Oracle Corporation for defining the user interface of a JavaFX[1] application.

Source root: The directory from which point on is strictly programming code. In this project’s case, only Java-files.

SOLID: A set of design principles, defining broad aspects of good object oriented design.

JavaFX: JavaFX[1] is a software platform for creating and delivering desktop applications, as well as rich Internet applications (RIAs) that can run across a wide variety of devices.

2 System architecture

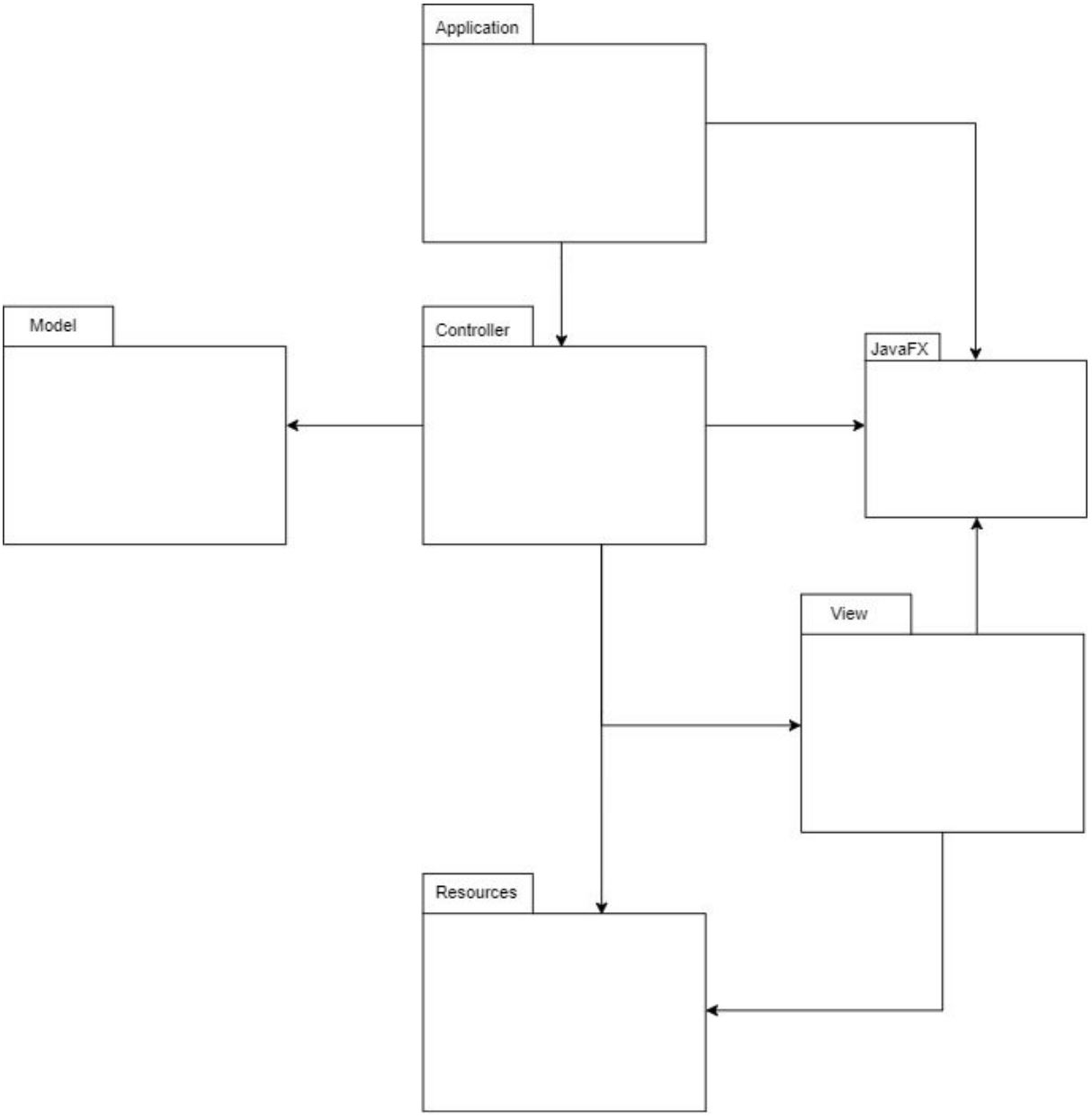

Figure 1. Package diagram[2]

The dependency between Controller and Resources comes from Maven[3] in combination with FXML-files. An FXML-file is most often considered a view but Maven does not allow non .java files to exist inside the source root meaning the FXML files cannot be included in the View package. If they were in the View package the Controller would not have any connection to the Resource package.

At a high level the program architecture was designed with design principles in mind. Most prominently were the “SOLID” principles considered. Coherency and independence were constantly taken into consideration when implementing components, this meant following the “High-Cohesion, Low-Coupling” principle which resulted in a code with less complexity and better readability.

The program was designed with extensibility and maintainability in mind, meaning following the well known “MVC” pattern to make sure that the model was without external dependencies so that it could potentially be reused.

To further increase re-usability of the code, classes were constantly grouped into logically coherent components and the structure was built trying to resemble reality.

2.1 Model

The model package holds classes representing the logic of the application, ergo, each individual game. Seeing as most of the games use continuous motion of some sort each game’s model is equipped with an update function that updates the state of the game.

Each game’s model also holds a list of IRepresentable, which represents an object that should have visual representation.

2.2 View

The view package holds classes that are used for the visual representation of elements from the model package and takes use of the library JavaFX[1] in order to accomplish this.

2.3 Controller

The controller package holds classes and interfaces that are used to manage views in view package and to handle user input such as key presses. In short, supply model with user input and relay information from model to view in order to visually represent the games. The controller package also takes use of JavaFX[1] as a timer for updating and drawing the games and as a way to relay which view MainApplication should be showing.

2.4 Application

The application package holds the class MainApplication which is responsible for starting the game by connecting it’s stage to the ViewNavigator and then telling the viewnavigator to load the start menu scene.

2.5 Resources

The resource package holds resources or data that can be accessed by the code. Images, icons, fonts and FXML files can all be found in the resources package.

3 System design

Design model
The design model can be found as an attached pdf file (see appendix).

Façade pattern:
The Facade pattern is used in package froggermodel within the model package where FroggerFacade acts as the interface that other packages can interact with.

Singleton pattern:
The Singleton pattern is used in the controller package, namely in ViewNavigator to prevent the creation of multiple instances of ViewNavigator which in turn would lead to the creation of multiple instances of the same controllers, models and views.

MVC(A):
MVC(A) is used to structure the entire project. The application starts the program, model holds game logic, controller relays info from model to view and provides user input information for model and view gives a visual representation of the program.
For more detail see chapters 2.1 - 2.4.

Module pattern:
Like MVC(A) the Module Pattern helps give structure to the project, dividing classes that are related in some way and organizing them into packages. This goes further than MVC as there are subpackages within each MVC-package. These have been divided by what game they are related to. Meaning all game logic classes associated with the game Frogger are located in the package froggermodel, SpaceInvaders in package spaceInvadersModel and so on. The one exception is the package common which contains interfaces and classes that multiple games use in order to avoid code duplication.

Factory pattern:
The Factory pattern is used in package froggermodel with the class LaneFactory whose sole purpose is to create different kinds of instances of Lane for FroggerModel.

4 Persistent data management

The project does not store any persistent data.

In regards to storage of resources the project has a maven[3] specific resource package outside of the source root where resources such as fxml-files, images and icons are stored.

5 Quality

In order to test the program the testing framework JUnit[4] was used. This made it easy to test functionality without a graphical interface initially. The tests are located in a seperate folder `test` where all games have their own testing file with test testing their functionality.

If any irregularities were found in the source code this would be brought up during weekly meetings in order for the responsible person to correct their mistake.

6 References

[1] JavaFX. [Online]. Available: https://openjfx.io/, Accessed on: 2020-10-23.
[2] Diagrams.net. [Online]. Available: https://app.diagrams.net/ Accessed: 2020-10-22.
[3] Maven. “Welcome to Apache Maven”. [Online]. Available: https://maven.apache.org/#, Accessed on: 2020-10-23.
[4] JUnit, “JUnit - About”, 2020. [Online]. Available: https://junit.org/junit5/, Accessed on: 2020-10-23.

7 Appendix

A pdf file named “DesignModel” can be found in the same folder as this file.