anOtherGame

System Design Document

Version 1.0
2017-05-08

Miranda Bånnsgård
Amanda Jonsson
Maja Nyberg
Allex Nordgren

This version overrides all previous versions.

1 Introduction

This is a document that describes the construction of The Lost Kitten application specified in the documents and analysis document.

1.1 Design goals

We have been developed the application towards some design goals.

- The design must be testable, which means that it should be possible to isolate parts for testing.
- The controller and graphical user interface should not be dependent on the model. They should be exchangeable and open for future development.

1.2 Definitions, acronyms and abbreviation

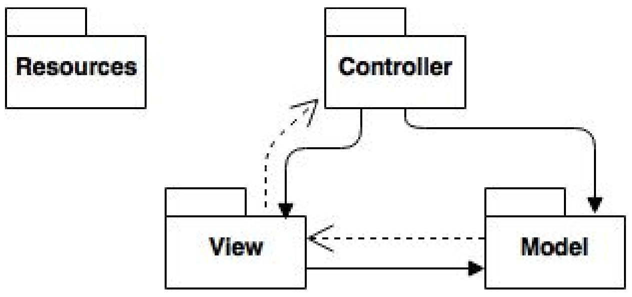
- MVC - Model-View-Controller pattern. This design pattern is used to separate concerns of the classes. The model contains the logic, the view represent the visualization of the application and the controllers controls the data flow and translates the user's interaction with the view into actions which the model then will operate.[1]

- JavaFX - A software platform for developing desktop applications.[2] It is graphics packages and media packages that enable the developer to design, create, test and debug.[3]

See the RAD for definitions.

2 System architecture

The application is a desktop application and it run on a single computer.
The application is divided into four top level packages. The arrows represent the dependencies.



[1] Design Patterns-MVC, https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm, 2017-05-16
Java SE Application Design With MVC, http://www.oracle.com/technetwork/articles/javase/index-142890.html, 2017-05-16
[2] JavaFX, https://en.wikipedia.org/wiki/JavaFX, 2017-05-15
[3] What is JavaFX?, http://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm, 2017-05-16

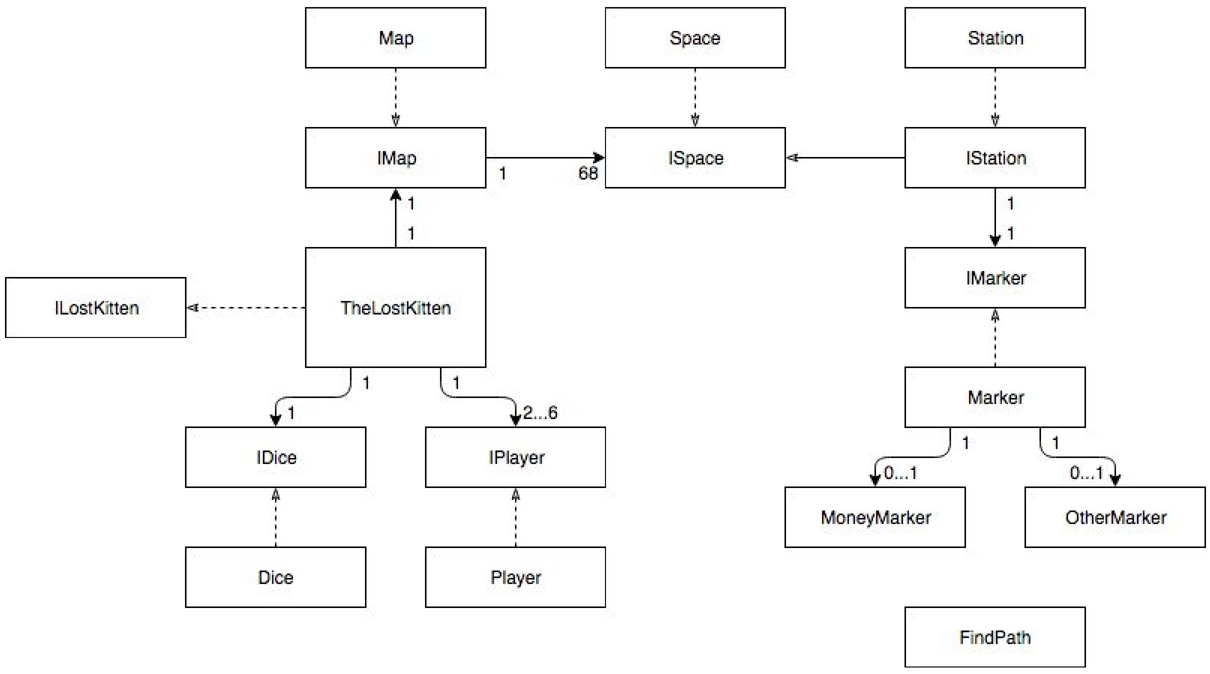
Resources - contains mostly pictures which is used in the application.
Controller - contains all the controllers.
View - contains fxml-files and some view classes representing the GUI.
Model - contains the object-oriented model, the logic.

The controller package has references to both View and Model. The Model package has no references to either of the Controller or View packages. The interfaces and abstract classes should be open for extensions and closed for modifications, and thereby implemented according to the open-closed principle.

3. Subsystem decomposition

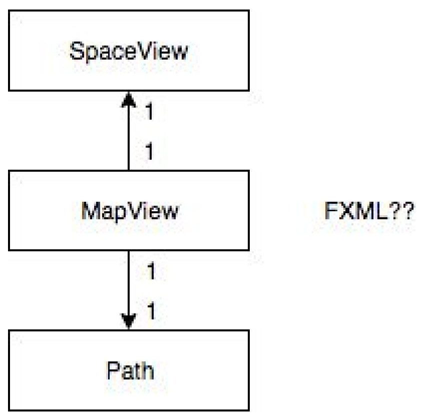
3.1 The Model

The Model package's design model:



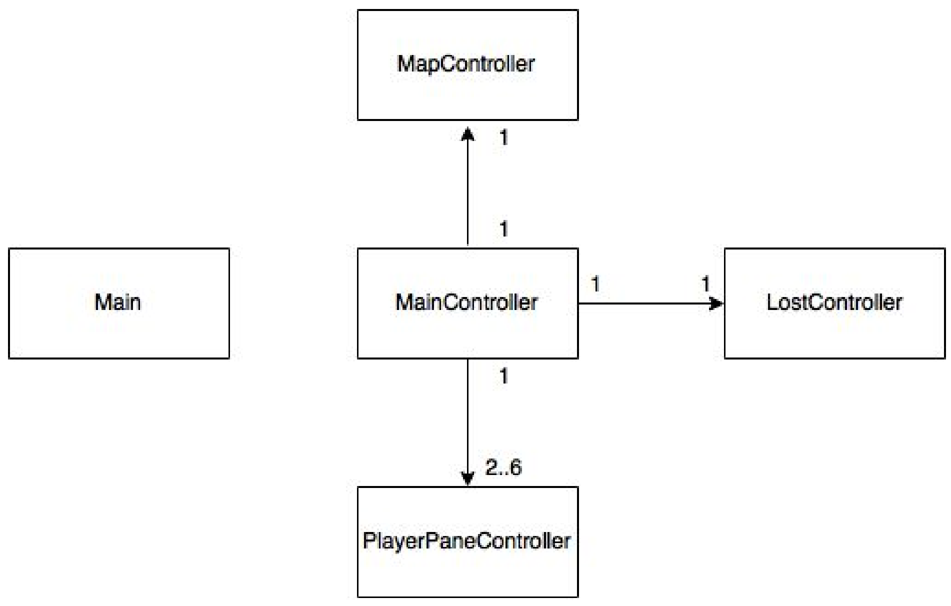
3.2 The View

The View package's design model:



3.3 The Controller

The Controller package's design model:



Recap: What is this doing (more detailed)
Divide it into top level subsystems. An UML package diagram for the top level. Describe responsibilities for each package (subsystem). Describe interface. Describe the flow of some use case inside this software. Try to identify abstraction layers. Dependency analysis Concurrency issues.

If a standalone application

- Here you describe how MVC is implemented

- Here you describe your design model (which should be in one package and build on the domain model)

- A class diagram for the design model.

else
- 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 ...

3.2 "...next software to describe" ... As above....

4. Persistent data management
The resources i.e. images are stored in the folder *Resources* in the *Src* folder for the game.

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

5. Access control and security
Different roles using the application (admin, user, ...)? How is this handled?

6. References