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

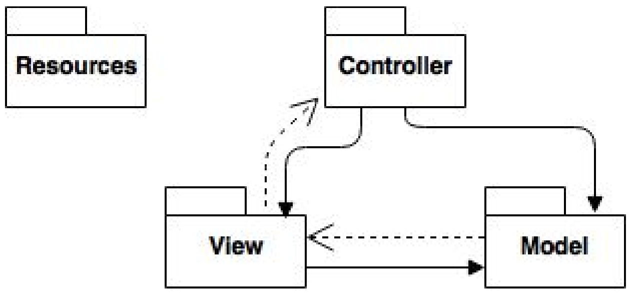
- MVC - Model-View-Controller pattern. This design pattern is used to separate concerns of the classes. The model contains the logic, the view represent the visualization of the application and the controllers controls the data flow and translates the user's interaction with the view into actions which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

2.1 General observations

The aggregate class

We have an aggregate class which is the TheLostKitten class. This class is responsible for the game logic and handles a player's turn. All calls from the controllers will pass through the TheLostKitten class.

Interfaces

We have been implemented against interfaces. We have seven interfaces, almost one per class in the model. This is a way to make the code open for extension but closed for modification. This is the design pattern open/closed-principle.

Unique JavaFx names

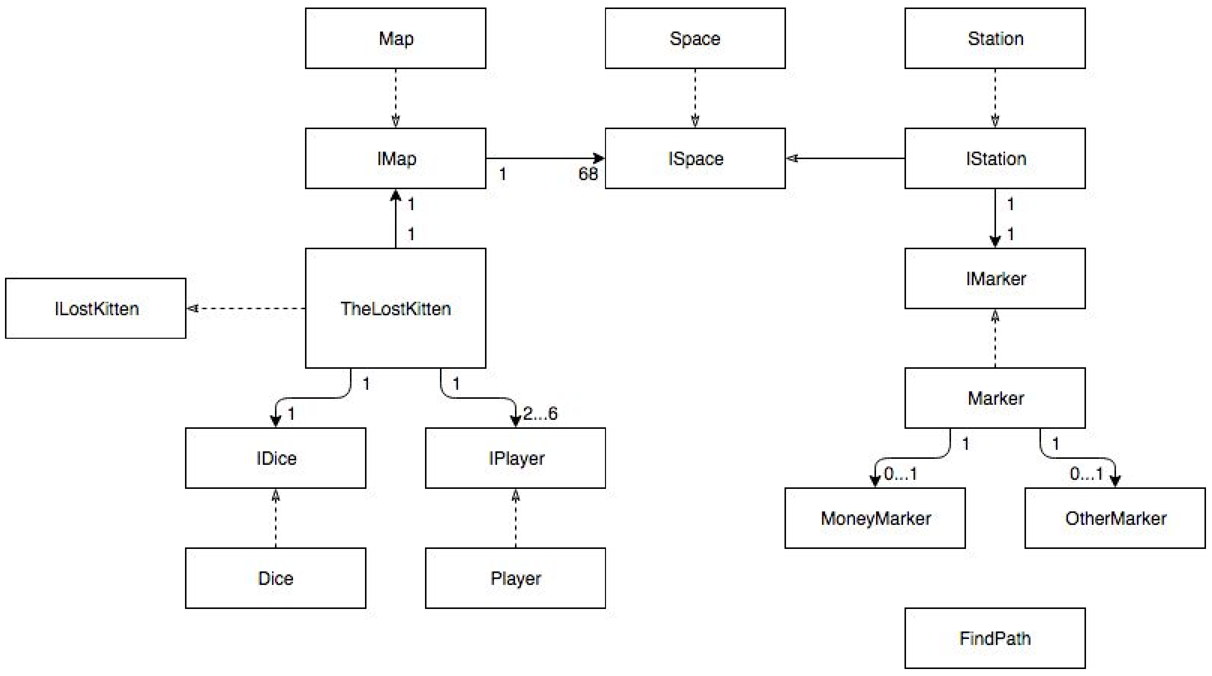
All labels, textfields and buttons for example needs to have unique JavaFX id, because they are referred to in many classes.

2.2 Dependency analysis

3. Subsystem decomposition

3.1 The Model

The Model package design model:

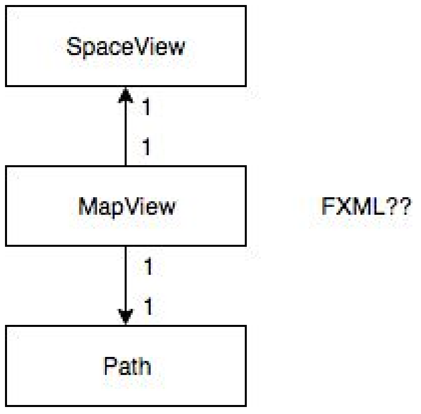


The Model, as mentioned further up, contains the object-oriented model which is the logic of the game.

Since our design must be testable we have implemented a Test-class for each class in the model. This means that we have the tests: DiceTest, FindPathTest, MoneyMarkerTest, OtherMarkerTest, PlayerTest, SpaceTest and StationTest. Each test is testing all the methods that are implemented in the class that is tested. These tests can be found in the Test-package, anOtherGame/tests/Model.

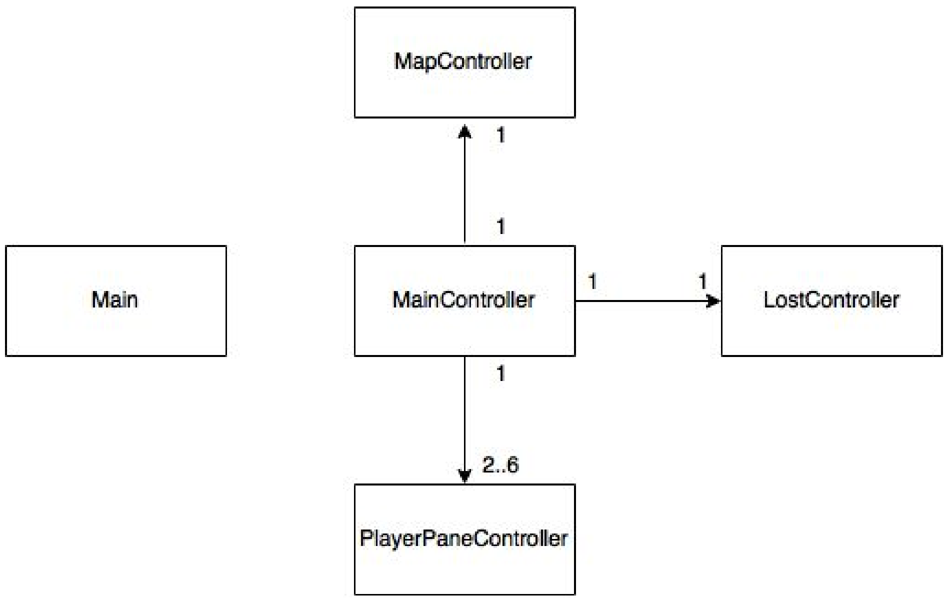
3.2 The View

The View package design model:



3.3 The Controller

The Controller package design model:



3.4 Model-View-Controller

This project uses the design pattern Model-View-Controller (MVC). The model is acting like a bridge between the controllers and the view. The user of the application will activate, by for example pushing a button, a controller. Subsequently the controller will invoke a method in the model and by that change state. The model itself do not undertake actions, it only approves commands from the controllers and do what the method should do. For example our TheLostController, which is a controller, invokes the method setNewBudget in the TheLostKitten class , which is a model class, to update the a players budget. The method setNewBudget are using a method from another Model class to increase or decrease the balance.

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.

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 …

4. Persistent data management

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

5. Access control and security

NA. This application is handled at the same way if you are a user as if you are the developer. It is launched by running the application and exited by either clicking on the 'Close Game'- button or clicking on the red cross on the window.

6. References

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

JavaFX, https://en.wikipedia.org/wiki/JavaFX, 2017-05-15

What is JavaFX?, http://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm,
2017-05-16