System design document for Operation 5A (SDD)

Table of Contents

Version: 1.0
Date 2013-05-23
Authors

Linus Hagvall, Vidar Eriksson, Martin Calleberg, Jonatan Magnusson

This version overrides all previous versions

1 Introduction

1.1 Design goals

The goal is to make a loosely coupled application which will be able to support different GUI:s. We want an object oriented program which should be easy to add functions to, such as a multiplayer features. For usability see the RAD.

1.2 Definitions, acronyms and abbreviations

- GUI, the graphical user interface
- RAD, Requirements and Analysis Document
- MVC, Model View Controller

2 System design

2.1 Overview

The application will be designed after the MVC design pattern.

2.1.1 The model functionality

The model's functionality will be exposed by the main model classes GameModel and World, even though MainModel is the top model. Further functionality will be placed in separate model classes, like Player and Enemy. As players and enemies are both characters in the world they will share much functionality in the interface Sprite.

We will use factories where there are small changes between different objects, like Weapon and Enemy.

2.1.2 Unique identifiers, global lookups

The SettingsModel will be accessible from anywhere in the application as settings potentially can change values many classes are dependant on.

2.1.3 Internal representation of text

All text visible to the user should be localizable. Therefore all the view classes should use keys for the text they should display. By doing this we can switch language without changing hard coded values.

2.1.4 Event handling

When an object is added which needs to be displayed as an object on the screen the model will fire an event with a specific string as a key which tells the view to add a new view class of the object added.
Events will be fired by the GamePanel as well, this because the GameController will listen to various inputs from the game. An example of such an event is mouse movement which is translated to map coordinates before being sent with the event.

All possible event names should be declared in the class which fires the event.

2.2 Software decomposition

2.2.1 General

The application consists of these modules:

- Main - a class used for launching the application.
- model - holds all the game's data.
- controller - holds the controllers which control the game.
- view - handles all the rendering of the game. Holds all the GUI classes.
- IO (inputOutput) - loads and saves data.
- Resources - holds easy to change values.

See Figure 1.

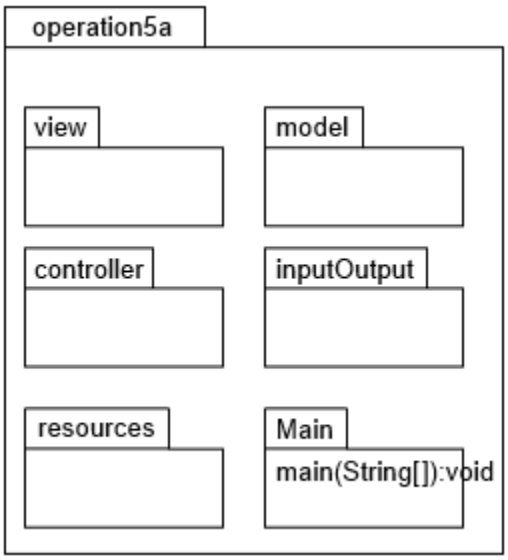

Figure 1: The different modules

2.2.2 Decomposition into subsystems

We will not decompose our system into subsystems.

2.2.3 Layering

The layering is as indicated in Figure 2 below.

2.2.4 Dependency analysis

Dependencies are as shown in Figure 2 below.

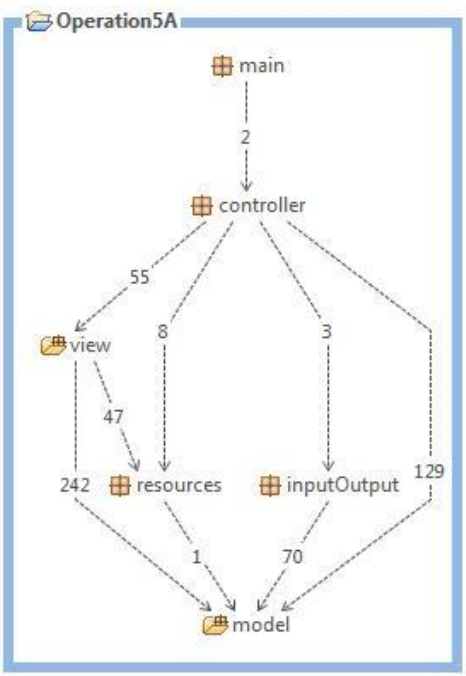

Figure 2: Layering and Dependency analysis

2.3 Concurrency issues

The application will be multithreaded, to some extent. However, the controller will be the single thread handling the data in the model, the other threads will be views repainting the data. Because of the single controller thread, there will hopefully be no issues with concurrency.

2.4 Persistent data management

The user will be able to save the current game session and resume it at a later time. All the high scores will be saved locally. We will use simple save files instead of databases.

2.5 Access control and security

There will not be any security concerns involving our application since no sensitive data is handled.

2.6 Boundary conditions

The application will be launched and closed as a normal desktop application. However, as we want the application to be able to be displayed in full screen, there will be close options in menus.

3 Implementation details

3.1 Swing

We chose Swing as our graphics library as we all were familiar with it and is advanced enough to handle our simple 2D graphics. We knew we would not need advanced animations so we did not need any fancy libraries to handle them, and could instead write them ourselves.

3.2 Dividing the code

During the project the project all members of the group has contributed with code for the models and controllers of the game. Big parts of the project have not had a specific person with full responsibility, but some parts have been more or less taken care of by a single group member.
This is how the project was divided:

- Jonatan has been responsible for items, both its code and graphics.
- Vidar has been responsible for threading the program. And designing the menu and it's submenus.
- Linus has been responsible for the pathfinding as well as a lot of the update methods.
- Martin has been responsible for creating the World, its components and graphics.

References

APPENDIX

- Appendix 1a - class diagram, world package
- Appendix 1b - class diagram, sprite package
- Appendix 1c - class diagram, item package
- Appendix 2a - sequence diagram, player move
- Appendix 2b - sequence diagram, player shoot