System design document for Röjarna

Contents

Version: 1.0

Date: 19/3-14

Author :

This version overrides all previous versions.

1 Introduction

1.1 Design goals

It should be possible to use a custom design for the game without having to modify the actual code. The GameBoard class should be very versatile, meaning that it should contain functionality not necessarily needed for this application, but can be used to create a different game than Röjarna.

1.2 Definitions, acronyms and abbreviations

- GUI, graphical user interface
- Java, platform-independent programming language
- Square, one geometrically square button located on the gameboard which the player can press and beneath which there can be a mine, a number or nothing
- Gameboard, a grid of squares in a rectangular shape
- Round, an entire playthrough of one gameboard in Röjarna ending either in a loss or a win
- Mine, an item which will be hidden beneath a square on the gameboard which will have a special effect on the game depending on the game-mode
- Game-mode, the game uses two game-modes (Classic and Campaign) which both will use the same basic rules for manipulating the gameboard but might extend the regular minesweeper gameplay in certain ways

2 System design

2.1 Overview

The game will be using the MVC design-pattern for coding where each model will represent a different game-mode of minesweeper.

2.1.1 Model implementation

An abstract class is used to define the most basic methods and also provide a number of abstract methods to be implemented by subclasses. These subclasses will each define a certain game-mode for the game, although being restricted to using an instance of the Gameboard class as an internal representation of the grid-of-equals being the viewable board. The model is implemented in such a way that all of the rules for what will happen when a round is playing will be handled there. For example, what will happen when you press on a a mine needs to be handled in these classes since each model represents a game-mode and these will handle mines differently.

There are also an interface for the powerups used in the campaign game-mode, making it easy to implement additional powerups at a later time.

2.1.2 Gameboard

There are two classes which collaborate to handle the internal (non-viewable) representation of what the minesweeper board looks like. One class (Gameboard) simply organizes the other (Square) in a 2D-array which is the size of the actual board in the game and provides different methods to manipulate these squares. Each square-class has one enum-class which is used as a state-indicator for each square on the gameboard. This enum has values such as flagged, meaning the player has right-clicked and put a flag on this square, or mine, meaning this square is a mine.

2.1.3 Rules

The classic game-mode will use the same rules as Microsoft Minesweeper, see references for these rules.

The campaign game-mode is designed to be a series of levels, where each level is one round of the classic game-mode.

Here, instead of an increasing timer in classic, a decreasing timer will be used which means the player will have a time-limit to complete each round. After one round has been completed (and resulted in a win) the player will be presented with a new round which has an increase in difficulty. This round will also have a time-limit, where the time will be a set amount for that round and adding the time which the player had left in the last round.

During these round the player will have the option of using different kinds of powerups, all of which will reduce the remaining amount of time the player has to complete the current round. One will allow the player to select an unclicked square and, without losing a life, reveal or flag that square depending on if it was a mine or not.

2.2 Software decomposition

2.2.1 General

Package diagram. For each package an UML class diagram in appendix

2.2.2 Decomposition into subsystems

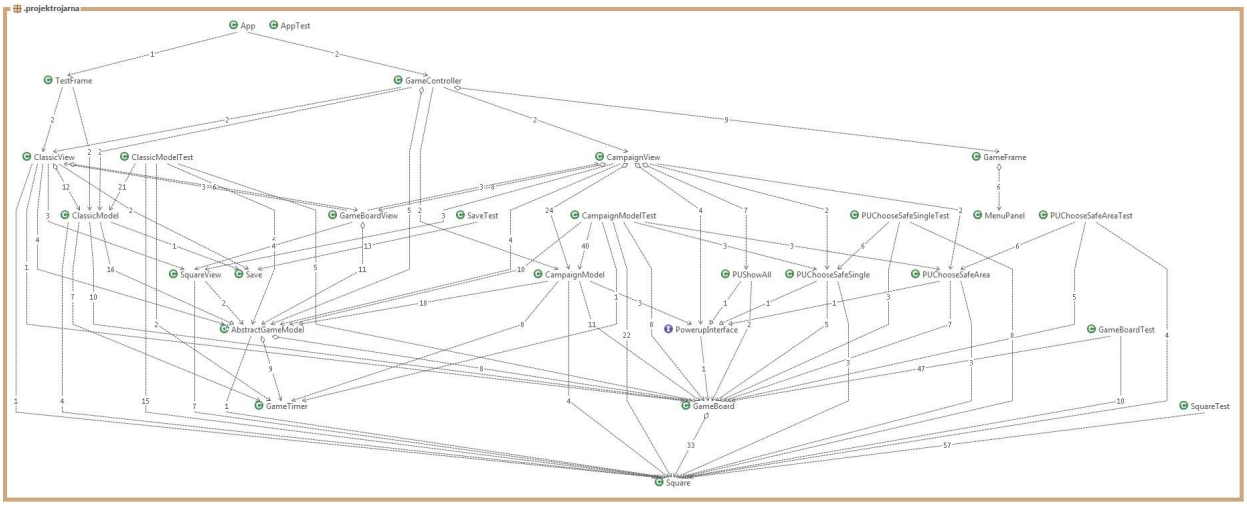
There is a single class handling IO (input/output) operations in the project. These operations are very simple as they only either read or write short lines of text in a file.

2.2.3 Layering

See figure below.

2.2.4 Dependency analysis

See figure below.



2.3 Concurrency issues

NA. This application runs in a single-threaded environment using the Swing thread. There should not be any concurrency issues.

2.4 Persistent data management

A class called “Save” will manage any data that needs storing. The current use of this class is very simple, as the only things stored are highscores (the 3 best ones) in the Classic game-mode.

2.5 Access control and security

NA

2.6 Boundary conditions

NA. The application launches and exits as a normal desktop application.

3 References

Minesweeper rules & history
https://en.wikipedia.org/wiki/Minesweeper_(video_game)

APPENDIX
Klassdiagram för paket