System design document for the Blockster project (SDD)

Version Blockster .. last iteration...
Date 2014-04-08
Authors Joel Tegman, Emilia Nilsson, Eric Bjuhr, Oskar Jönefors
This version overrides all previous versions.

Contents

1. Introduction

1.1 Design goals

The design will use the MVC design with possibility to change framework. The model will be independent from the view and controller, which will implement the LibGDX framework.

“ The design must be loosely coupled to make it possible to switch GUI and/or partition the application into a client-server architecture. The design must be testable i.e. it should be possible to isolate parts (modules, classes) for test. For usability see RAD “

1.2 Definitions, acronyms and abbreviations

All definitions and terms regarding the core Monopoly game are as dened in the refer-ences section.
GUI - graphical user interface.
Java - platform independent programming language.
JRE - the Java Runtime Environment. Additional software needed to run an Java application.
MVC - a way to partition an application with a GUI into distinct parts avoiding a mixture of GUI-code, application code and data spread all over.

2 . System design

2.1 Overview

The application will use a MVC model.

2.1.1 The model functionality

Our model will implement different interface which will make up the fundamental structure of the game.

“ The models functionality will be exposed by the interface IMonopoly. To avoid a very large and diverse interface the functionality will be split into interfaces for Player and Board. IMonopoly will be the top level interface acting as an entry to other interfaces IPlayer and IBoard, see Figure. “

2.1.2 Rules

The rules will be taken care of in the model.

“ The rules of the game could vary. This could be handled by different implementations of affected classes (sub classes). Yet, here we have chosen a different approach. All rules

will be been re-factored to a Rules class. Model classes delegates the rules-dependent parts to the Rules class.
In this way the rules also can easily be used to enable/disable components in the GUI. "

2.1.3 Unique identifiers, global hook-ups

We will be using a util package that will be accessible from anywhere.

" We will not use any globally unique identifiers for any entity. There will be no look ups from anywhere in the application (objects will be directly connected or accessible without an identifier). Example: The spaces-objects will not be stored in any global accessible data structure to be looked up. They will be directly connected to interacting objects (GUI, etc.). See also Spaces. "

2.1.4 Blocks

Blocks will make up the entire map and will be manipulated in different ways.
Block is an interface with multiple properties.

" To have a uniform handling of spaces (possible configurable), all kinds of spaces are kept in a single list. The ordering of the spaces is determined by the ordering of the list. This will make it easy to create different views of the spaces (just traverse list and create a view for each space). "

2.1.5 Event handling

" Many events, state changing or not, can happen during the play (new player, dices equal, go to jail, etc.). A need for a flexible event handling is evident. If this is done at an individual level i.e. each receiver and sender connects, we possible end up with a hard to understand web of connections (also possible many receivers for one event/sender). How and when should connections b e set? Also, during testing of the model we possible would like to disable the event handling.
To solve the above we decide to develop an event-bus. All connections of senders/receivers and transmitting of events is handled by the event-bus.
The connections could be setup at application start for static parts. Dynamic parts must have means to connect to the bus at any time. "

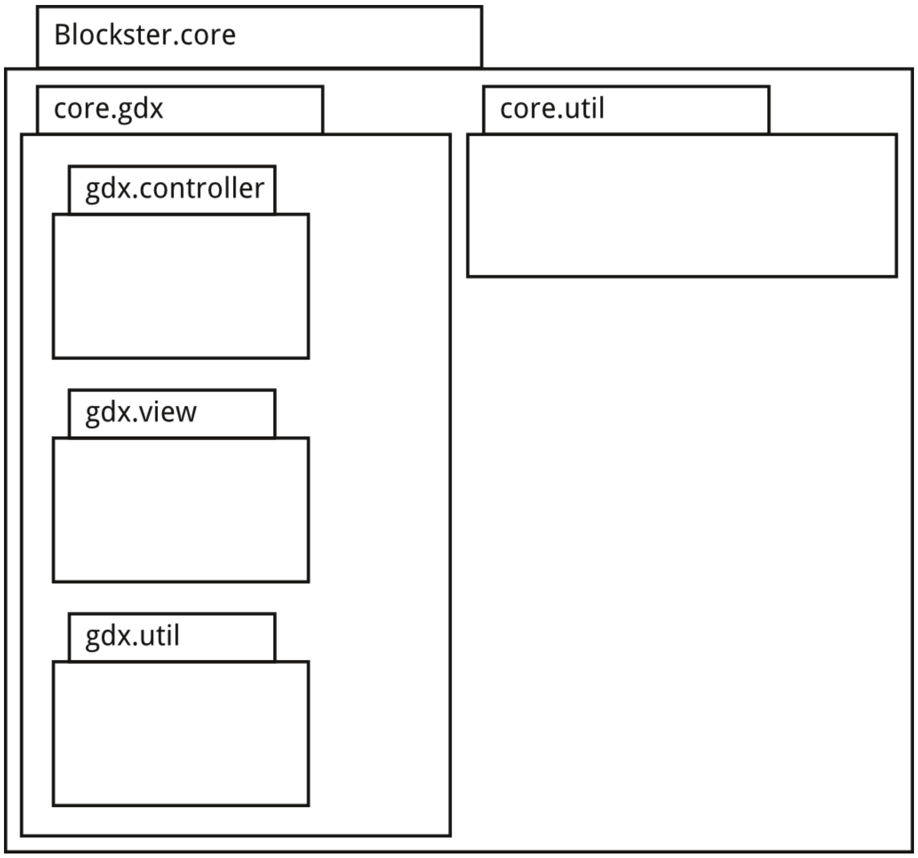
2.1.6 Internal representation of text

" All texts should be localizable. Therefore internally all objects will use language in-dep endent keys for the actual text. Using the key the object can retrieve the actual text. "

2.2 Software decomposition

2.2.1 General

- Core, the main structure of the game, interfaces, the model part of MVC
- Core.gdx, contains the gdx application
- Core.gdx.controller, handles the input control from the user
- Core.gdx.view
- Core.gdx.util
- Core.util, handles the directions and collision.



“ The application is decomposed into the following modules, see Figure 2.
dialogs, GUI dialogs to interact with users. View parts for MVC.
view, main GUI for application. View parts for MVC.
eventbus, classes for the eventbus.
ctrl, is the control classes for the MVC model
adapter, contains an EventAdapter for the model. Broadcasting events for model state changes
model, is the core object model of the game. Model part of MVC.
Main is class holding main-method, application entry point.
io, is for le handling. “

2.2.2 Decomposition into subsystems

“ The only subsystem is the le handling in package io (not a unified subsystem, just classes handling io). “

2.2.3 Layering

“ The layering is as indicated in Figure below . Higher layers are at the top of the gure. “

2.2.4 Dependency analysis

“ Dependencies are as shown in Figure. There are no circular dependencies. “

2.3 Concurrency issues

There is only one active thread in this application, however, the user input is event based and we thus have to take this into account when creating the controller. We have solved this by setting a volatile flag according to user input, and upon each rendering of the game we check the flag for user input and then change the model accordingly.

NA. This is a single threaded application. Everything will b e handled by the Swing event thread. For p ossible increased resp onse there could b e background threads. This will not raise any concurrency issues.

2.4 Persistent data management

A file for storing data which indicate the user’s progress.

“ All persistent data will be stored in flat test files (format, see APPENDIX). The files will be;
A file for spaces. The ordering of the spaces is used as the internal (implicit) ordering for the spaces-objects. This ordering will b e directly reflected in the GUI. See further directions in RAD.
Localization les containing entries (texts) for the text keys in the application. “

2.5 Access control and security

NA

2.6 Boundary conditions

NA. Application launched and exited as normal desktop application (scripts)

3 References

LibGdx: http://libgdx.badlogicgames.com/
http://libgdx.badlogicgames.com/nightlies/docs/api/
MVC: http://sv.wikipedia.org/wiki/Model-View-Controller