System design document for the Kelde, the wrath of Ra's al Ghul project (SDD)

Version: 2.0

Date 2015-06-29

Author Philip Tibom, Anders Bolin, Hossein Hussain, Daniel Olsson

This version overrides all previous versions.

1 Introduction

1.1 Design goals

The design must be modular, strictly using the MVC-pattern and use clean code. The GUI and Controls must be easily replaced in order to be ported to mobile devices. For usability see RAD.

1.2 Definitions, acronyms and abbreviations

- GUI, graphical user interface.
- Java, platform independent programming language.
- JRE, the Java Run time Environment. Additional software needed to run a Java application.
- Host, a computer where the game will run.
- 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.
- libGDX, a Java cross platform game engine.
- Android, operating system for mobile devices.

2 System design

2.1 Overview

The application will strictly use MVC. We will use event buses in some situations for example when distribution collision detection events to the affected objects in the game.

2.1.1 The game world

The game will contain a world which holds and modifies all interact able objects. The UseCase "New game" will create a new object (world) that contains the map, player, monsters, treasures etc. We have chosen to make these objects into models where we

expose only the necessary information with interfaces such as IEntityPlayerKelde where we can get or set the current health value of the player.

2.1.2 Polymorphism

We have chosen to have a common interface for each monster, each item etc that can collide with other entities. In such way we can in a very clean and few lines of code use polymorphism to distribute collision events to the appropriate objects.

2.1.3 Reflection

One of our main goals has been to make the game as modular as possible. For example we wanted a game designer to be able to insert new monsters, objects, edit the map, expand the map and so on from a drag and drop editor. To enable this dynamic way of creating objects we are reading a map(text) file with all the generated information from the chosen map editor. And we create objects dynamically with reflection based on the given information. Thus the game gets very modular and future development of the game will be incredibly fast.

2.1.4 Event handling

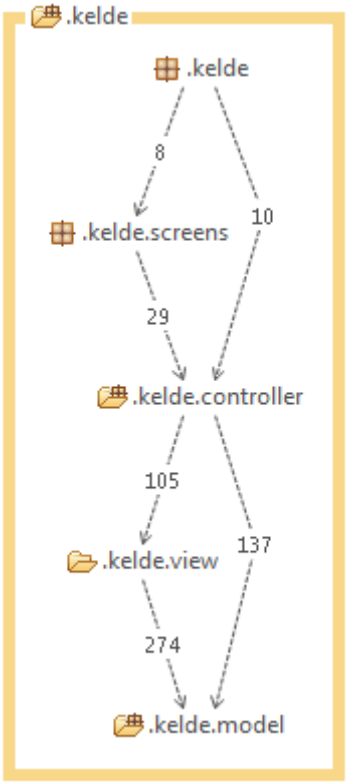
In order to remove dependencies we are using our own event buses to distribute events. For example when the player collides with a treasure. We want to let the treasure know that it should open up and drop items on the ground. So we do this by sending an event from collision detection system.

2.2 Software decomposition

2.2.1 General

The application is decomposed into the following packages:

- Controller
- Model
- View
- Screens



The screen package contains the launchers for the application. The actual game loop so to speak.

The controller package contains 11 packages:

- Entities
- Events
- Gameworld
- Guioverlay
- Intro
- Inventory
- Items
- Physics
- Services
- Startmenu
- Worldobjects

The Model package includes the following 11 packages:

- Constants
- Encapsulation
- Entities
- Gameworld
- Guioverlay
- Intro
- Inventory
- Items
- Physics
- Startmenu
- Worldobjects

The View package includes 10 packages:

- Entities
- Gameworld
- Guioverlay
- Intro
- Inventory
- Items
- Physics
- Startmenu
- Ui
- Worldobjects

Essentially everything in our game has a corresponding Model, View and Controller class for maximum modularity. The controllers handles system calls, the view handles the drawing and the model handles data such as the current position in the coordinate system or the current health value.

Each object is updated on every iteration of the game loop. So everything is initiated in a nice hierarchy.

2.2.2 Decomposition into subsystems

We do not have any subsystems.

2.2.3 Layering

Our application is too extensive to describe this in words and to include a picture on a single page. Please visit our github repository to see the full image.

https://github.com/ptibom/Kelde/raw/master/Documents/SDD/layers.png

2.2.4 Dependency analysis

There are no circular dependencies. Please see the same image in our github repository.

https://github.com/ptibom/Kelde/raw/master/Documents/SDD/layers.png

2.3 Concurrency issues

NA. This is a single threaded application. Everything is handled by the libGDX library. For example the game sound is managed by libGDX secretly in its own thread. But this is nothing that will be exposed to the programmer.

2.4 Persistent data management

All persistent data will be stored in flat text files.

2.5 Access control and security

This is a single-player game and thus security is not important. If they player wants to cheat and hack the game it is OK! It could even be fun!

2.6 Boundary conditions

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

3 References

1. MVC, see http://en.wikipedia.org/wiki/Model-view-controller

APPENDIX

NA, appropriate images is already inserted or linked to.