System design document for Chicken Invasion

Version: 1.0

Date: 20/04-2016

Author: Albin Bååw, Kristoffer Ek, Pedram Shirmohammad

This version overrides all previous versions.

1 Introduction

1.1 Design goals

1.2 Definitions, acronyms and abbreviations

2 System design

2.1 Overview

In this section we explain the overall design choices.

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

2.2.3 Layering

2.2.4 Dependency analysis

1 Introduction

1.1 Design goals

The design must be loosely coupled to make it possible to switch GUI. The design must be testable i.e. it should be able to isolate some parts (classes) for test. For usability see RAD.

1.2 Definitions, acronyms and abbreviations

All definitions and terms regarding the core Monopoly game are as defined in the references section.

- GUI, graphical user interface.
- Java, platform independent programming language.
- JRE, the Java Runtime Environment. Additional software needed to run an Java application.
- Throwable object, an object that the player can throw.
- Enemy, a target the player can throw the throwable object at.
- 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 MVC model with a small adjustment. Because of the architecture in libgdx we choose to have the controller and view all together.

2.1.1 Aggregates

The application will consist of a single class aggregate with root class ChickenInvasion. All calls from other parts of application to model will pass through the ChickenInvasion class

2.1.2 The model functionality

The model's functionality (API) will be exposed by the ChickenInvasion class. To avoid a very large and diverse interface to the class there are methods to access other classes in the aggregate (sub API).

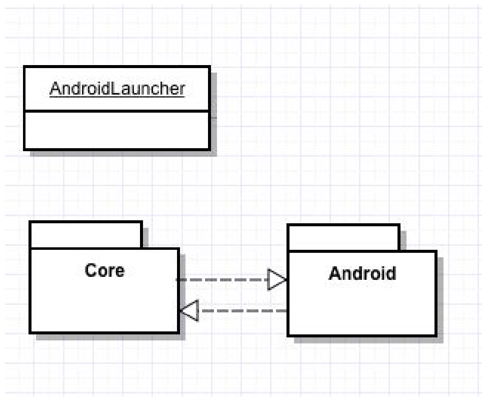
2.1.3 Enemies

Enemies are handled uniformly by the wave class. ChickenInvasion class will get a list of enemies to draw from the wave class.

2.2 Software decomposition

2.2.1 General

The application is decomposed into the follow top level packages (arrows for dependencies)



- AndroidLauncher is the application entry class.
- Core, the OO-model
- Android, androidspecifik classes.

2.2.2 Decomposition into subsystems

The only service (implemented as a subsystem) is the core package.

2.2.3 Layering

N/A

2.2.4 Dependency analysis

There are no problems with the dependencies.

2.3 Concurrency issues

N/A

2.4 Persistent data management

All data except images are saved as Shared Preferences on the users Android device. The images are saved as images normally are saved on the device.

2.5 Access control and security

N/A

2.6 Boundary conditions

N/A. Application launched and exited as normal Android application.

3 References

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

APPENDIX