­System design document for Chicken Invasion

Version: 1.1

Date: 14/08-2016

Author: Albin Bååw, Kristoffer Ek, Pedram Shirmohammad

This version overrides all previous versions.

# Contents

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

2.3 Concurrency issues

2.4 Persistent data management

2.5 Access control and security

2.6 Boundary conditions

3 References

APPENDIX

# 1 Introduction

## 1.1 Design goals

The design goals of this application is to be loosely coupled to the frameworks used by the application. This implies that the models can’t have any obvious framework dependencies. We have decided on the other hand to let the views and controllers depend on frameworks. A modified version of “Model View Controller”-pattern will be used.

The application must always be testable, this means that we have to be able to isolate some parts(classes) to be able to test them.

For usability See RAD.

## 1.2 Definitions, acronyms and abbreviations

* GUI - graphical user interface.
* Java - platform independent programming language.
* Model View Controller – a design model used to partition an application with a GUI into distinct parts.
* State pattern – a design model where a object has different states, and it’s behavior varies based on which state it’s in. The state can change based on internal and external actions.
* Singleton – A singleton-class is a class which there can only be one instance of and that one instance can be accessed globally.
* Throwable object - An object that the player can use to defeat enemies.
* Enemy - a enemy the player has to defeat to not lose the game.

# 2 System design

## 2.1 Overview

The application will use multiple interfaces to separate the models from the frameworks used. These interface will be used by the model to get access to framework classes and methods, the interfaces will be implemented by classes using the framework.

The application will use a modified Model View Controller. Because of the architecture in a used framework called Libgdx, we must have both the controller and the view in the same class.

The application will also use a State pattern to differentiate the Running-state, Stopped-state etc.

# 2.2 LibGDX

LibGDX is the framework used by the application. The framework adds functionality that helps drawing and physics needed to create the application. We will never use the framework in the models, but the classes used by the model may use them, with out the model knowing.

## 2.3 Design patterns

## 2.3.1 State pattern

For all of the game states we use a design pattern called state pattern. RUNNING, STOPPED, PAUSED and GAMEOVER are all states that are handled differently in the contoller depending on the current state in the model.

## 2.3.2 MVC pattern

## The GameModel class are completely unknowing about the libGDX framework and the controller. Which creates a separation of responsibilities in the code. This makes the code more reusable and easier to maintain. 2.3.3 Abstraction

All the classes in the Helpers and Enemy\_Throwable packages Implements an interface to create a project were it is easy to extend and add similar functionality to the game.

## 2.4 Software decomposition

## 2.4.1 General

The Invasion package contains sub packages for both LibGDX framework adapters, reusable game components, but also the game itself. All classes are grouped by their functionality in the following packages.

* Helpers
* CallBacks
* Weapons
* CIStore
* EnemyThrowables

2.2.2 Decomposition into subsystems

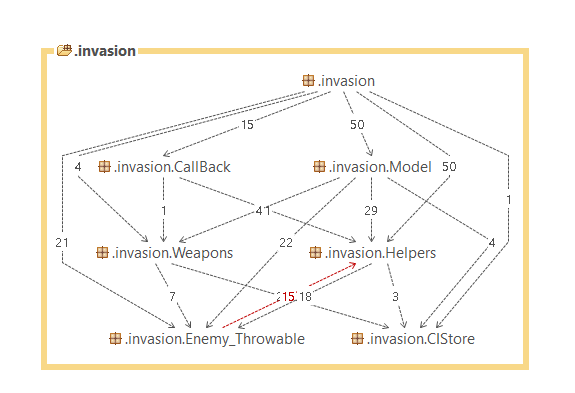
The only service (implemented as a subsystem) is the core package.

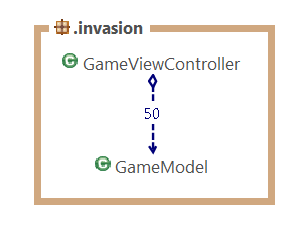
## 2.2.3 Layering

N/A

## 2.2.4 Dependency analysis

There are no problems with the dependencies.

The model is completely separated the Contoller and the View. 



## 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