System design document for Illegal Aliens

Oscar Carlsson[1] — Emil Josefsson[1] — Kevin Rudnick[1]
Johan Svennungsson[1]

[1] *Objektorienterat programmeringsprojekt, civilingenjörsprogrammet i informationsteknik, Chalmers tekniska högskola*

May 2017

1 Introduction

This is the System design document (SDD) for the application ”Illegal Aliens” and will describe and explain the system aspects of the application. The document is divided into sections, where each section describes its corresponding system aspect.

1.1 Definitions, acronyms and abbreviations

MVC Model-View-Controller, used for separating and dividing the application into three parts.

Object-oriented design System is built on separate objects to solve its purpose

High cohesion - low coupling Components work together to solve their modules responsibility and lower dependencies between modules

Circular dependency Two classes depend on each other to be functional

Package (Sub)folder with classes

1.2 Design goals

These are the major design goals we strive to implement in the application.

- Overall application design using the MVC pattern
- Object oriented design
- High cohesion (related code close to each other) - low coupling (few dependencies between classes and modules)
- Flexible design: easy to extend and add new parts/features and easy to change something without affecting something else
- Domain driven design: use the domain model as a starting point during the development of an application

2 System architecture

Overall top-level description of our system.

2.1 Overview

The game will be object-oriented and written in java and utilize the Libgdx framework. It will implement the MVC design pattern (active MVC) between models, views and controllers. The game is primarily made for desktop use but since it'll be built on `libGDX` porting the game, to say mobile devices, won't be difficult.

2.2 Garbage collecting

A class should be written to handle and collect all drawables. This allows dynamically adding and removing all stages and sprites that have to be rendered on the screen in one place.

Since Java has an automated garbage collector, the rest of the objects that should be disposed will be handled by it. In some places, we put the object's reference to null to decrease the time taken to dispose it.

2.3 Map

The game will feature different maps that can be chosen by the player in the mainmenu. The different maps to play will be chosen in a MainMenu-state, that then will switch to a Game-state loaded with the specified map. The maps will be built by nodes forming a grid. The enemies will be able to navigate their way to the Whitehouse through the grid using Dijkstra's algorithm.

2.4 Enemies

Enemies will be split into smaller classes representing different types of enemies. They will all inherit an enemy class. The enemy class will have MVC but also a factory since the game will spawn a lot of enemies.

2.5 Towers

The towers classes are almost split in the same way as enemies, as described above. Each tower has its own class (mostly containing tower-specific variables and objects), which all extend a common abstract super-class. Methods in the super-class are inherited by the sub-classes, such as `shoot`, `target` and so on.

The towers have their own independent MVC, which handles most of the logic for the towers and their view representation.

2.6 Projectiles

Projectiles will be fired by towers to kill enemies. Projectiles will be an own class with sub classes for different types of projectiles. Projectiles will be created in towers shoot-method, however after they are fired they will have nothing to do with the tower that made them. Different projectiles will be able to implement interfaces giving them further abilities over the regular projectile. For instance, there will be an area of effect projectile that causes damage to all enemies inside a certain area on impact.

2.7 Screens

Each screen consists of multiple Stages. `MainMenuScreen` holds all Stages related to Main Menu, such as `MapSelectStage`. `GameScreen` holds all Stages related to Game, such as `RightGameUIStage`.

2.8 Stages

Each Stage hold all objects related to UI such as `Image`, `ImageTextButton`, `TextButton` as provided by the libGDX-framework. A controller class that `extends ClickListener` is injected and listens for `InputEvent`s made by the Player.

2.9 RoadManager

Roadmanager is responsible for managing everything with the road. Most of all the calculation regarding the road is however done in helper classes such as DijkstraSolver. The roadmanager is used by BoardObjects to make sure that no BoardObjects can be placed onto the road. It also gives aliens their path on the road network, calculated by DijkstraSolver.

2.10 IASprite

IASprite is a modified Sprite that improves the reliability of adding, changing and removing textures dynamically. It contains method for rotating towards a `Node` and getting the angle from itself to a `Node`. It also changes the logic of the position, so it's in the middle of the sprite instead of the bottom-left corner.

2.11 Views

Views are used for showing objects added to the User Interface and `Map`. E.g. `BoardObjectView` listens on changes from `BoardObjectObserver`.

2.12 Texture handling

Instead of creating a new Texture for each object placed on `Map`, we use `TextureHandler`s with `private static final` attributes and `static` getters for each Texture.
The specific texture for e.g. a Bullet (Projectile) is then returned by `ProjectileTextureHandler`.

This benefits the program in two ways; all textures used for objects references to the same object (great for memory), textures can easily be added or changed in the same place and textures never have to be disposed (because of the `static` attribute).

2.13 Factories

Factories are used for creating objects which have to be created a lot, such as `enemies`. These classes and their methods are declared `static`, meaning they don't have to be created as object before being able to use them.

2.14 Models, views and controllers

The system has a number of models, views and controllers. Models handle logic, views handle graphical representation and controllers handle user input and the connection between models and views.

2.15 Hiscore

The project is bundled with an empty `SQLite` database `IllegalAliens.sqlite`.
The implementation for resolving to an existing database is slightly different depending on the system of choice e.g. `desktop` or `android`, therefore `DesktopDatabaseResolver` is only relevant for a Desktop-application. If the project would be ported to `android` an additional `AndroidDatabaseResolver` would need to be implemented.

3 Software decomposition

3.1 Package overview

`.controllers` Controller-classes part of MVC-pattern.

`.hiscore` Classes related to creating and accessing the bundled `SQLite` database `IllegalAliens.sqlite`

`.models` Model-classes part of MVC-pattern

`.models.boardobjects` All objects that can be placed on `Map`.

`.models.boardobjects.buildings` Buildings that does not shoot any projectiles, but affect enemies in certain ways, e.g. `Wall`.

`.models.boardobjects.towers` All different Towers that shoot `.projectiles`.

`.models.enemies` Different Enemies that use `.path` to move across the `.path.map`

`.models.enemies.levels` Classes for creating a level

`.models.enemies.waves` Classes related to creating the next wave of `.enemies`.

`.models.executive_orders` All different Executive Orders that mostly affect the monetary system.

`.models.politics` The Political System.

`.models.projectiles` All different projectiles that's being fired by `.models.towers`.

`.models.superpowers` High cost, large effect superpowers that can be used by the Player.

`.utilities.cooldown` Help classes to set different objects on cooldown, e.g. `.models.superpowers`.

`.utilities.path` Classes related to Enemies finding their path to target.

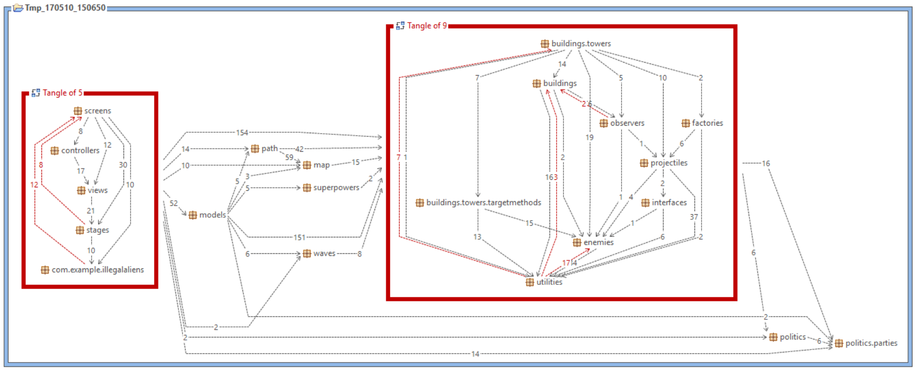
`.utilities.path.map` Classes related to the Map, such as `MapNode` with neighbors and `MapParser` for parsing a `.txt`-document

`.screens` Depending on the state of the Game, the correct Screen is shown, e.g. `GameScreen` during gameplay and `MainMenuScreen` when main menu is shown.

`.views` View-classes part of MVC-pattern.

`.views.stages` Different Stages used by `.screens`.

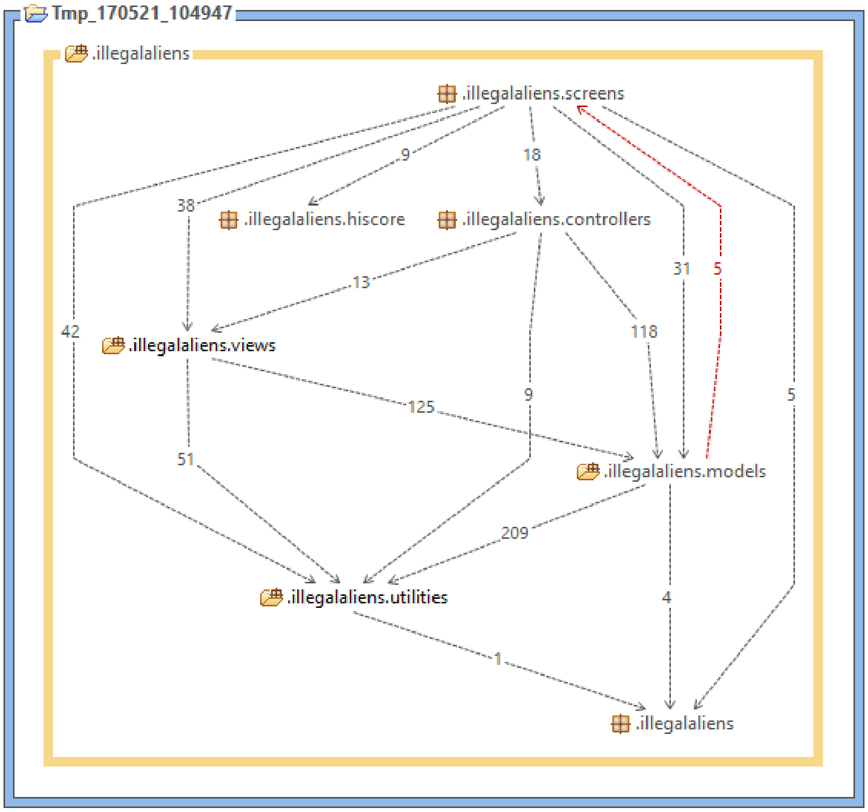
`.views.textures` Classes for handling `Texture`s used in the game.



Figur 1: Old package structure

3.2 Dependency analysis

`Figure 1` is an old overview of all dependencies and tangles in the system before refactoring. Almost none of this structure exists after later refactoring.



Figur 2: Final package structure

`Figure 2` describes how the structure of the packages look. More details in appendix.

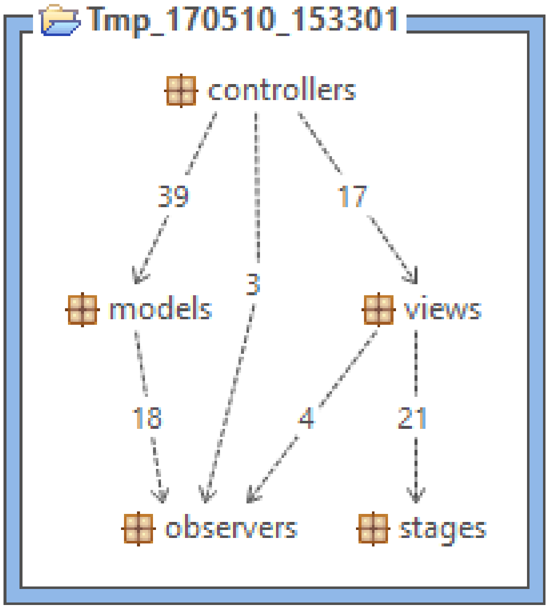
One of the goals during development has been to minimize all kind of unnecessary dependencies, by analyzing with STAN. Sadly, one circular dependency cannot be removed since it has its roots in the framework itself (LibGDX). The affected classes are Screen and Game. More information about the irremovable (without breaking the game) dependency can be found here.

https://gamedev.stackexchange.com/questions/67232/...

3.3 Layering

Package layering can be found in appendix.

3.4 MVC analysis



Our MVC is run as an active MVC, where the model communicates indirectly with the view(s) through an observer. The view uses stages to help build the UI (user interface). NOTE: The image above is only a representation of how our MVC as a whole works. It is not a image of how the package structure is.

4 Persistent data management

Our application does not currently support reloading older instances of the game, such as save games. The remaining persistent data such as images are stored in an assets folder in the project and are loaded and instantiated by the TextureHandler-classes when the application is launched. All external data have simple and obvious naming, most assets have the same name as their corresponding class.

To save the hiscore we have bundled the project with an empty SQLite database with a single table `hiscore` with the column `score`. On game over, the recent score (total enemies killed) will be added to the database. The Hiscore is then accessible from the main menu, where the top 20 highest score will be listed.

4.1 Tests

Tests have been made for highly logical classes containing lots of calculations. They have their own corresponding subpackage in `.src.test` package, matching `src.main` package structure.

5 Access control and security

Our application does only support one user (the player).

6 References

MVC:

https://en.wikipedia.org/wiki/Model\OT1\textendashview\OT1\textendashcontroller

Objectoriented Design:

https://en.wikipedia.org/wiki/Object-oriented_design

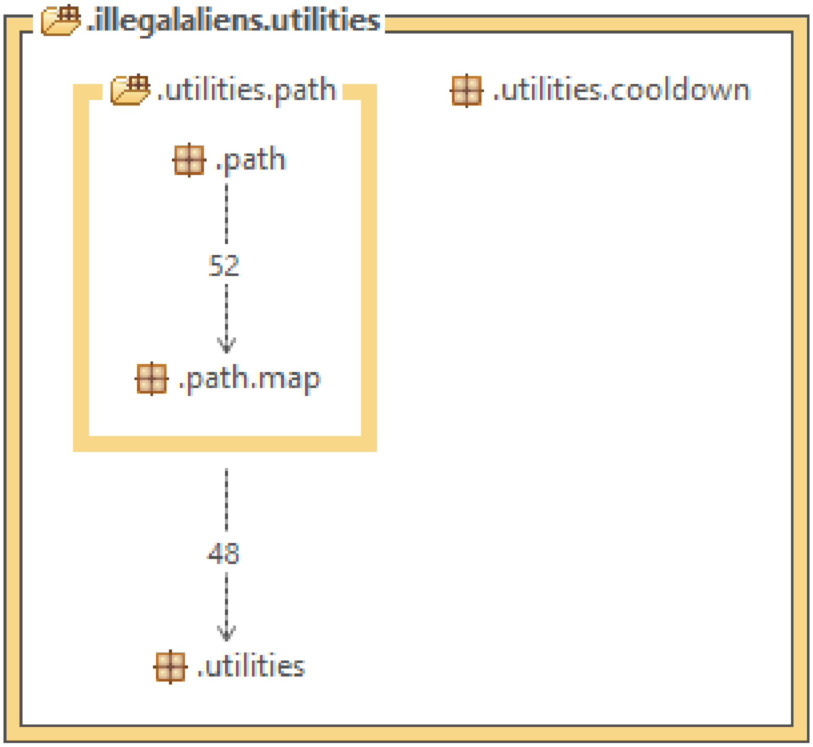
High cohesion, low coupling:

http://www.cse.chalmers.se/edu/course/DIT952/slides/4-2a%20-%20High%20cohesion,%20Low%20coupling.pdf

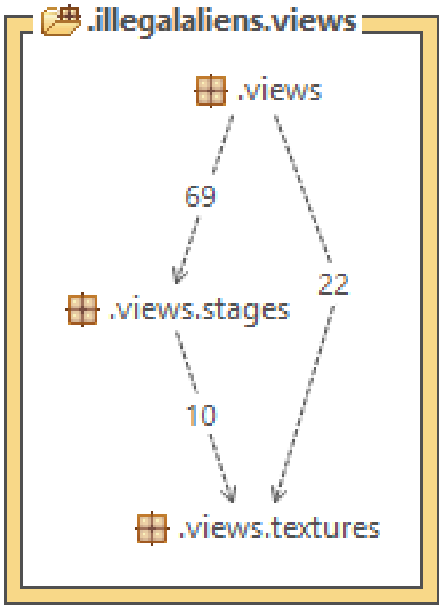
Domain driven design:

https://en.wikipedia.org/wiki/Domain-driven_design

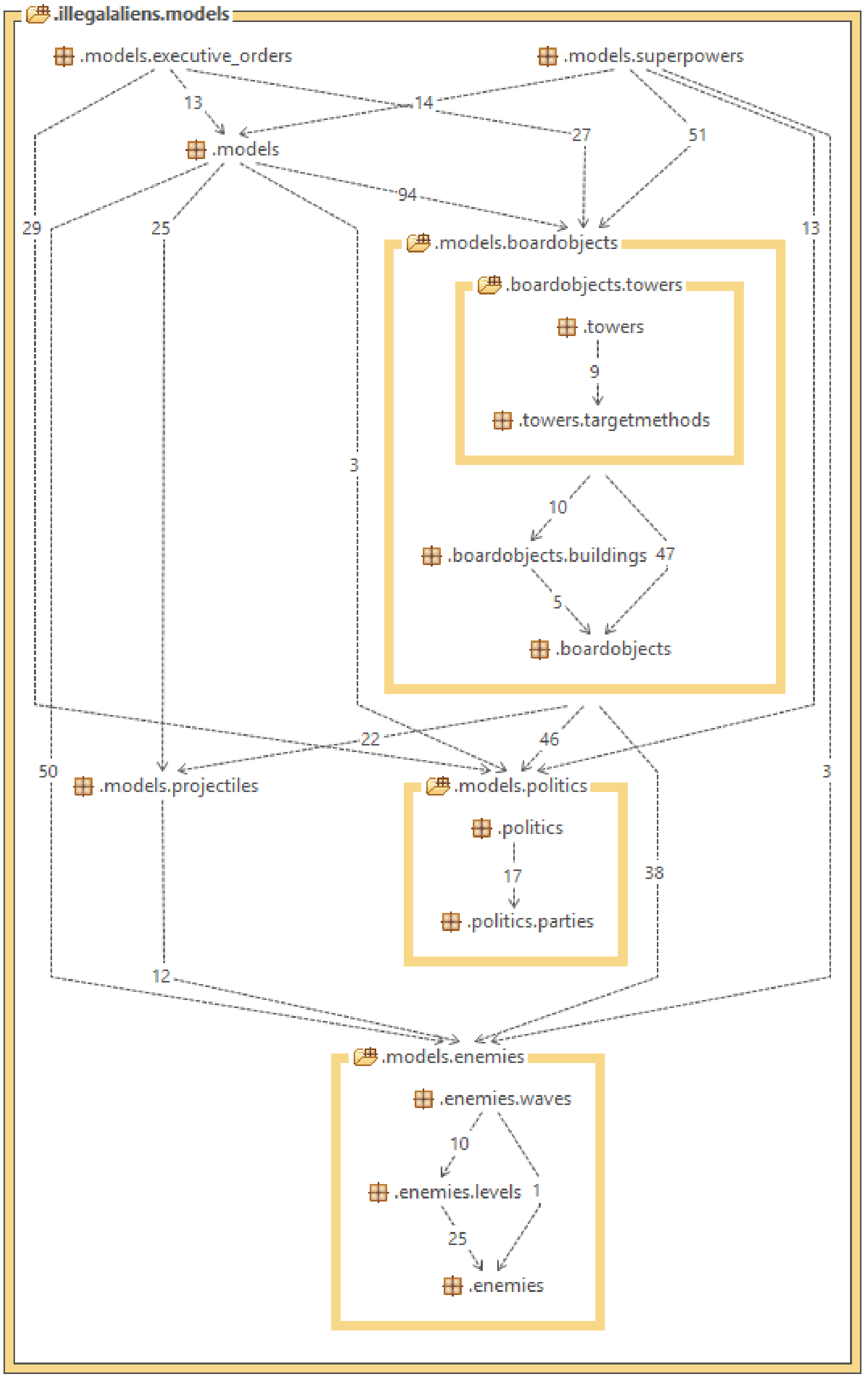
7 APPENDIX



Figur 3: Utilities structure



Figur 4: Views structure



Figur 5: Models structure