System design document for the “Programming United Network Game” project.

Version: 0.3
Date: 2016-05-04
Authors: Marcus Pettersson, Martin Sixtensson, Gustav Svensson, Erik Sänne

This version overrides all previous versions.

1. Introduction

1.1 Design goals

The design must partition the applications different parts according to the MVC-structure. The design should use the server-client structure to communicate between computers over a network. The design must be testable. See RAD for usability.

1.2 Definitions, acronyms and abbreviations

- GUI - graphical user interface.
- Java - platform independent programming language used in application.
- JRE - Java Runtime Environment. Software needed to run application.
- Host - The computer which starts a game.
- Server - Service provider.
- Client - Requests service from the server.
- MVC - Model-view-controller
- Tile - Construction block of levels in application. Decides how it looks and interacts with the game.
- Wave - Game term

2. System design

2.1 Overview

The application will use a MVC model. The application will use a server-client structure. To start a game the host must start a server. To join the game the players must then join the game as a client. As much as possible/feasible should be handled by the server.

2.1.1 Aggregates

The application will consist of two class aggregates since the application is divided into two parts; server and client.

2.1.2 The model functionality

Most of the model functionality will be on the server. The client's model will only serve as a model for the view, and pass on the user data to the server.

2.1.3 Map

Maps are created from map files. A map files contains a list/array of tile id's, telling how the map should be interpreted by the game. The difference between tiles are mainly cosmetic but some tiles such as wall tiles and light tiles are interpreted differently. Wall tiles are used for collisions within the game. Neither zombies or players are able to pass through said tiles. Light tiles function as lights in the game.

2.1.4 Unique identifiers, global look-ups

Players will be given a unique ID from the Server.

2.1.5 Event paths

Events are handled by standard java EventHandlers and the event chain goes GUI → Controller → Model which then calls the appropriate methods in Servercommunicator to send the data to the server.

2.2 Software decomposition

2.2.1 General

The application contains the following top-level packages.

- Launcher, application entry-point
- Client, the client sided portion of the game:
    - View, contains the applications different views. Since the game is very graphical, the views of different components of the application have been separated. I.e PlayerView, ZombieView, BulletView and so forth.
    - Model, the clients representation of the model.
    - Controller, controls applications process pipeline and event handeling.
- Server, the actual game model. Sends all calculated information back to the player:

    - serverUnits, server models.
    - serverWeapons, contains the games weapons and how they should behave.
    - serverWorld, contains map handler telling the server how to interpret different map tiles.
    - Unithandler, contains spawner class, creates enemy units
- Utilities, contains different help utilities such as a map loader and sound effect handler.

2.2.2 Decomposition into subsystems

Only included subsystem is mapeditor.

2.2.3 Dependency analysis

Analyses of dependencies within the application was created with software STAN.
No dependency issues are present between the top level packages(*see figure 1 and 2*).

There are though dependency issues in the server between serverWeapon and serverUnits packages since serverPlayer references serverWeapon which references to serverBullet serverBullet class from serverUnits. We have tried to resolve this issue but found no good way to solve it. *(see figure 3)*

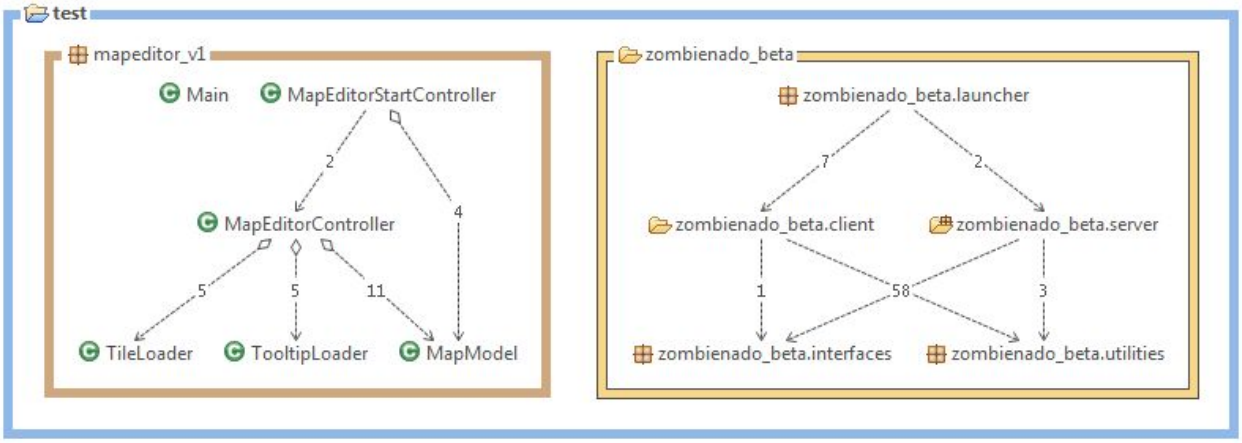

Figure 1: Dependency map over whole application

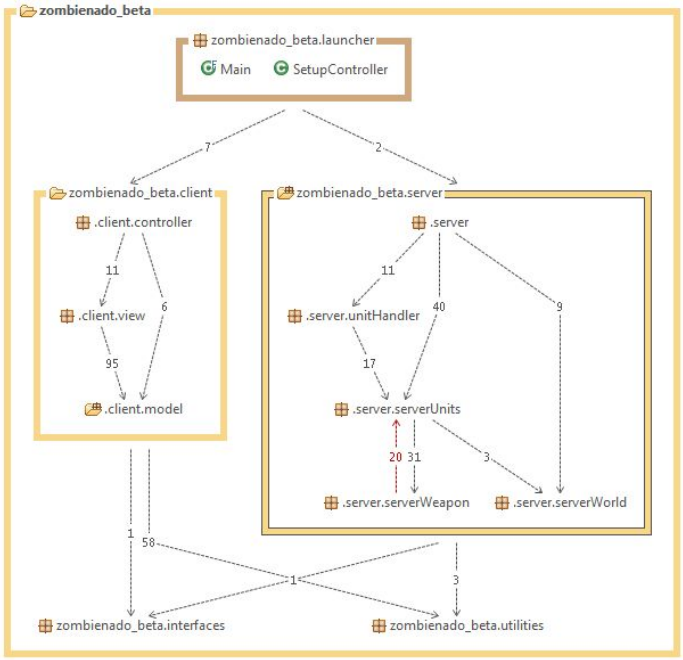

Figure 2: Detailed dependency map over package zombienado_beta

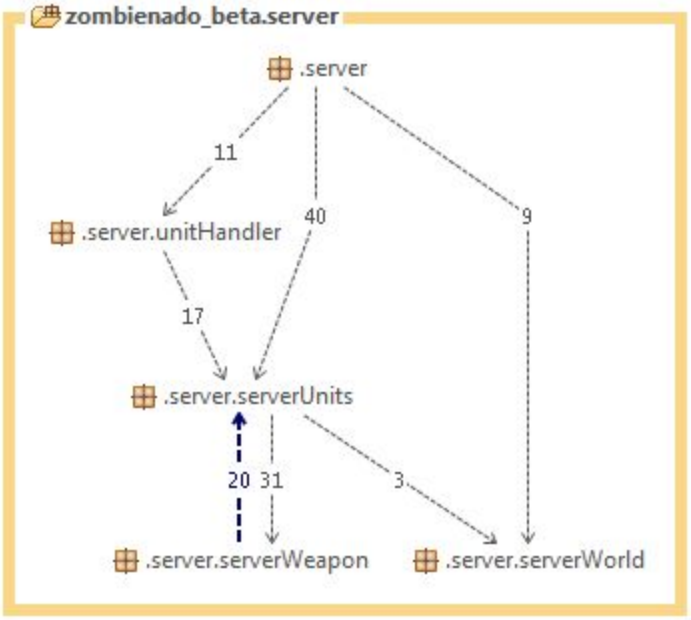

Figure 3: Dependency map over server package

2.3 Concurrency issues

NA

2.4 Persistent data management

Persistent data will be saved in text files. Only persistent data is map files which tell the application how to build the playable levels.

2.5 Access control and security

NA

2.6 Boundary conditions

Application can be exited and launched as normal application. However the application only launches a launcher which have to be used in order to start a server or join an ongoing game. To start a server a port number has to be entered, as well as prefered map. To join a game the hosts IP-address and port number has to be entered. The application can also exit itself when the game is lost.