System Design Document (SDD)

Tansk

Table of Contents

Version: 1.0
Date: 2013-05-26
Author: Victor Sandell, Erik Lundholm, Tobias Olausson, Christoffer Henriksson

1 Introduction

1.1 Design goals
The applications design must be suitable for a client-server implementation.

The design must feature a strong and indepedent game model. The design must be testable i.e. it should be possible to isolate parts (modules, classes) for test. For usability, see RAD.

1.2 Definitions, acronyms and abbreviations
- GUI, graphical user interface
- Slick2d, a library for game graphics and rendering
- Kryonet, a library used for implementing a client-server system
- Server, a computer or instance/state of the game where the game will run

2 System design

2.1 Overview
In this section we explain the overall design choices.

2.1.1 Heritage-structure
All objects on the map are based on an abstract class called Entity.

The inheritance continues with another abstract class called MovableEntity. That class is used by every object on the map that needs to be able to move.

The next stage is also based on abstract classes. In order to create for example different tanks, weapons and projectiles these are also built up with another abstract class each, containing the attributes they need.

2.1.2 Game conditions
When playing the game you might want to change the objective of the game, or perhaps how often weapons should spawn. The class GameConditions takes care of this, and when creating a new game, a new GameConditions object is created and the new conditions needs to be set.

2.1.3 The world
Every entity is stored in a world object. This class takes care of the collision detection and keeps track of every entity. World also contains objects that aren't entities such as the objects in charge of spawning tanks, powerups and weapons.

2.1.4 Game handling
The class that holds everything together is the GameController class. All of the other handlers are declared here. These handler classes take care of storing and accessing sound and images. There is also a handler that takes care of animations. The handling classes are essential for the application to function properly since sound and images needs to be accessible everywhere.

2.2 Software decomposition

2.2.1 General

See figure 1.

- controller, controller class
- event, event related classes
- gamemodes, different types of game modes available
- model, the core object model of the game
- network, network related classes
- powerups, power-ups (excluding weapons)
- powerups.pickups, graphical representations of power-ups
- resource, sound and rendering resources
- spawnpoints, spawning points
- states, game states (menus, views)
- tanks, different types of tank models
- terrain, walls and other obstacles on the map
- view, GUI components
- weapons, projectiles and turrets
- weapons.pickups, graphical representations of “pick-upable” weapons
- Tansk is class holding main-method, application entry point

2.2.2 Decomposition into subsystems

The application has a few sub systems. It features an Animation-, Sound-, and Image-handler, aswell as a MapLoader.

2.2.3 Layering

See “Figure 1: Dependency analysis and layering” down below for the layering.

2.2.4 Dependency analysis

Dependency is shown in figure below.

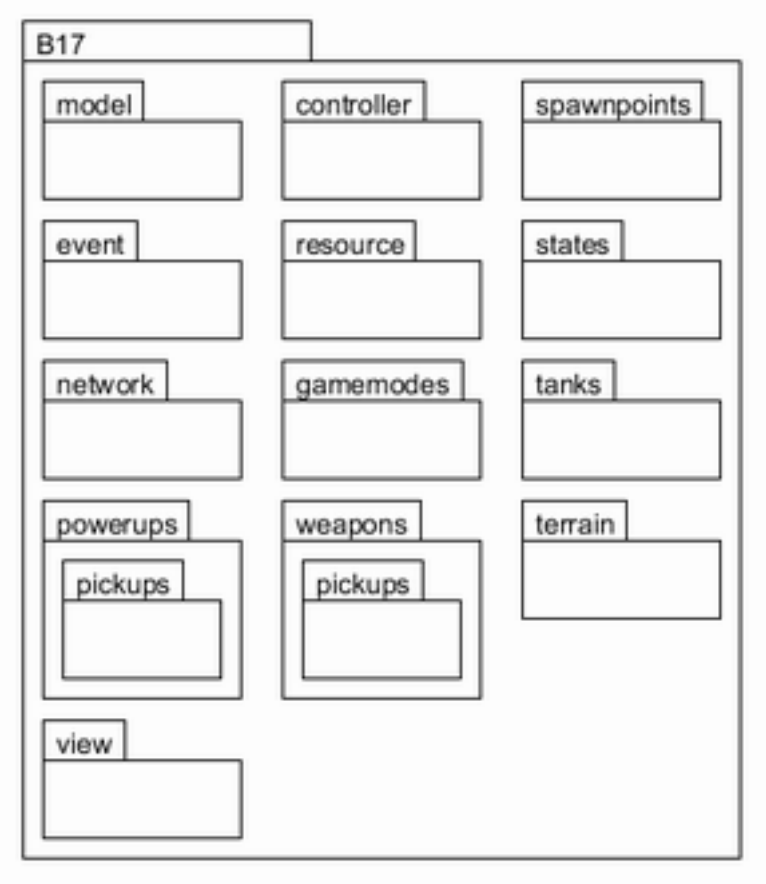

Figure 1: Packages

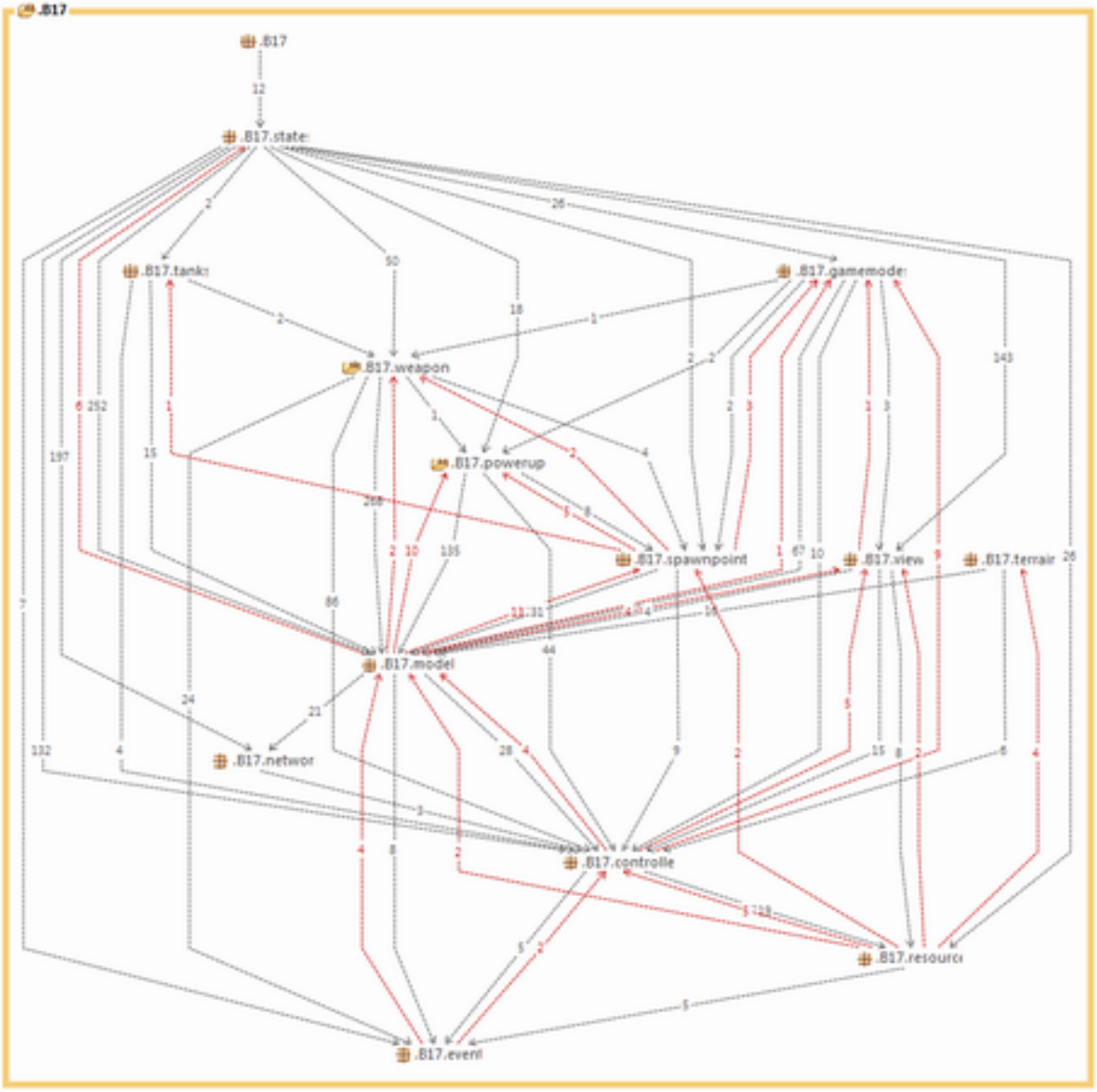

Figure 2: Dependency analysis and layering

2.3 Concurrency issues

The applications uses two threads. The game loop itself is one thread, and another thread is sending and receiving network packages. This leads to some concurrency issues that have been dealt with. There's also another concurrency issue in the game update loop. When the objects in the world are iterated and updated their data may change. This required a threadsafe concurrency-issue-preventing hashmap to be used instead.

2.4 Persistent data management

The player name is stored in a plain text file. This application does not use any other persistent data management.

2.5 Access control and security

NA

2.6 Boundary conditions

NA

3 References

Slick2D, http://slick2d.org
Kryonet, https://code.google.com/p/kryonet/

APPENDIX

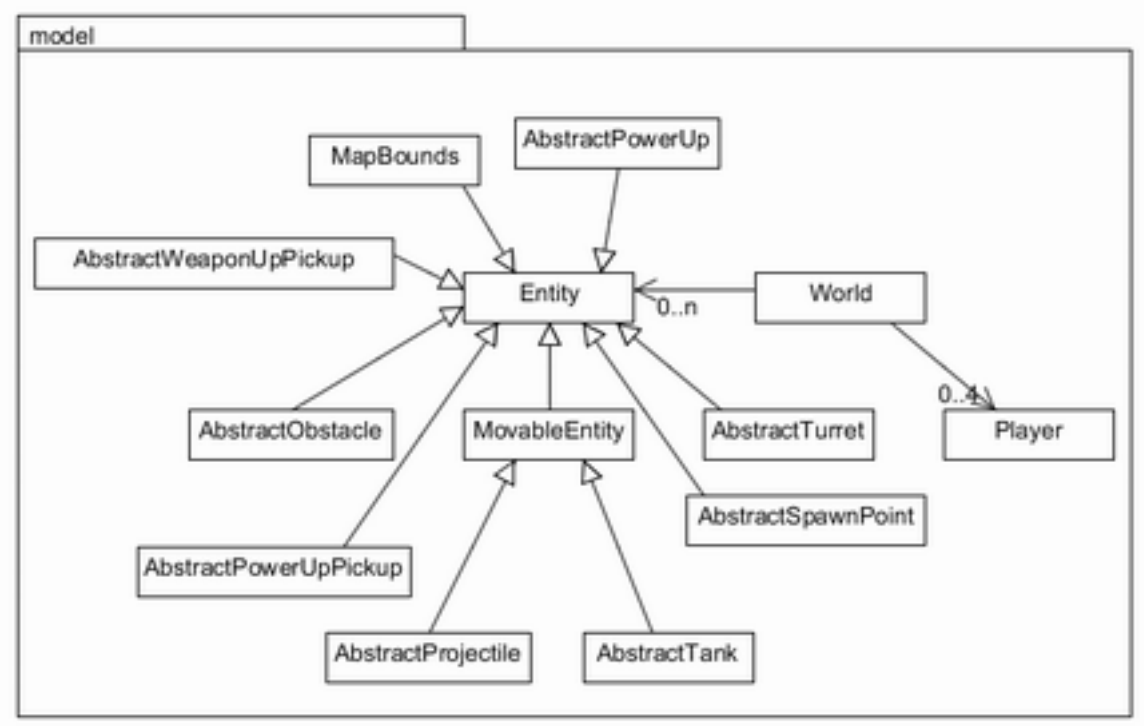

Figure 3: The model package

- AbstractObstacle
- AbstractPowerUp
- AbstractProjectile
- AbstractSpawnPoint
- AbstractPowerUpPickup
- AbstractWeaponPickup
- AbstractTank
- AbstractTurret
- Entity
- MapBounds
- MovableEntity
- Player
- World