System Design Document for Project Blast (SDD)

Table of Contents

Version: 1.0.0

Date: 2014-05-24

Authors: Anton Freudenthaler, Mattias Nilsen, Axel Savén Östebo & Alex Tao

This version overrides all previous versions.

1 Introduction

1.1 Design goals

The design of the code should follow the MVC pattern, it's especially important that the view is separated from the model so that another graphics library or set of graphics can be used without having to write any new game logic. The design should allow testing to be done for all major use cases. The code should be structured into separate packages with a common parent class. The rest of the code should as long as it's possible only interface with that parent class.

1.2 Definitions, acronyms and abbreviations

- GUI, graphical user interface
- Java, platform independent programming language
- 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 implements the MVC pattern
In this section we will explain the overall design choices.

2.1.1 The model

The model class, called BlastModel, is responsible for running the entire game. It has exposed some of its methods via the interface IBlastModel for use by the controller and view. BlastModel has a list of every object used in the game and is responsible for triggering their actions in response to certain events and removing them when they are destroyed.

2.1.2 Entity

Entity is an abstract class at the top of the hierarchy. Everything that's drawn on the screen has entity as its parent. It has a position a name and a collisionbox.

2.1.2 Hero

The playable heroes all have the Hero class as a common parent, it provides a unified interface for their primary and secondary attacks and some common functionality like ammunition and respawning.

2.1.3 Core

Cores are used by some heroes primary and secondary abilities when they don't have enough information to complete the task on their own. For example when a hero puts down a bomb that should explode in a certain pattern the explosion should stop when hitting certain entities. Only the model knows where all entities are so it uses a cores step and create system to accomplish this. When something is created by a core the following procedure is used.

1. Ask the core where it would like to place its next entity.
2. Call the method step on the core and pass the entity at that position, or null if there is none.
3. If step returns true it's safe to call create on the core if it returns false check if the core is done.
4. Repeat this process until the core says it's created.

2.1.3 Powerup

The powerup class handles boosts in power, speed and ammunition. It uses the strategy pattern. One applied powerup equals one step faster, one bomb more or one more square of range. This class can both apply power and also negate itself. This means that if you apply and negate an equal amount of times it will be as if nothing happened.

2.1.4 Hazard

This is something dangerous. If you walk into this you will die, and if something explosive touches it it will explode instantly. The most common hazard is Explosion. Hazards cannot be destroyed. They either stay in one place forever or die off on their own. None of these move but they can be created instantly in great numbers.

2.1.5 Explosive

This is a bomb, a fist, a fireball or a drone. Basically something that blocks players paths, and explodes in a cross shape after a while, killing players who get in the way. These bombs can be pushed by the Brute, and Drones can move by themselves. Fireballs fly forward on their own too. Apart from this, explosives generally remain stationary. You can stand on your own explosive after you deploy it, but you can’t reenter it if you leave.

2.2 Software decomposition

2.2.1 General

The application is decomposed into the following modules.

- View, the main GUI for the application. This is the view part for MVC.
- Model, this is the core object model of the game. This is the model part for MVC.
- Control, this is the control object for the game, it handles input. This is the control part for MVC.
- Test, the test package contains all JUnit tests for the application.

2.2.2 Decomposition into subsystems

Not applicable.

2.2.3 Layering

Layering is indicated in figure 1 below.

2.2.4 Dependency analysis

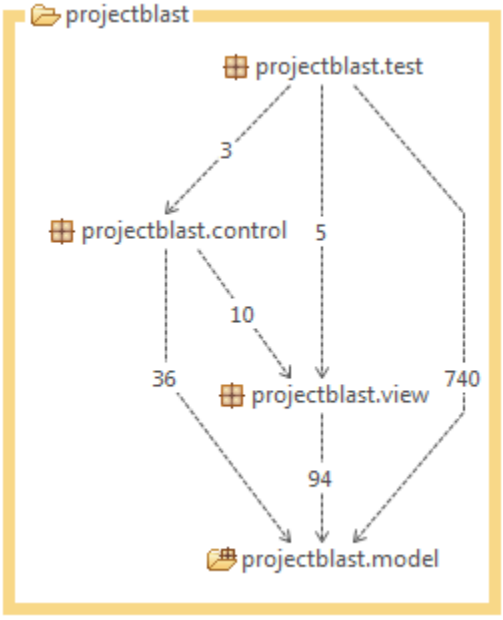

Figure 1. Visualisation of dependencies between packages

2.3 Concurrency issues

Not applicable. The application is run in a single thread.

2.4 Persistent data management

Not applicable. No persistent data is saved or read by the application.

2.5 Access control and security

Not applicable.

2.6 Boundary conditions

Not applicable. Application is launched and exited as a normal desktop application.

3 References

http://slick.ninjacave.com/javadoc/ - Javadoc API for Slick2D
http://www.mapeditor.org/ - Map Editor Tiled, used for map design

Appendix

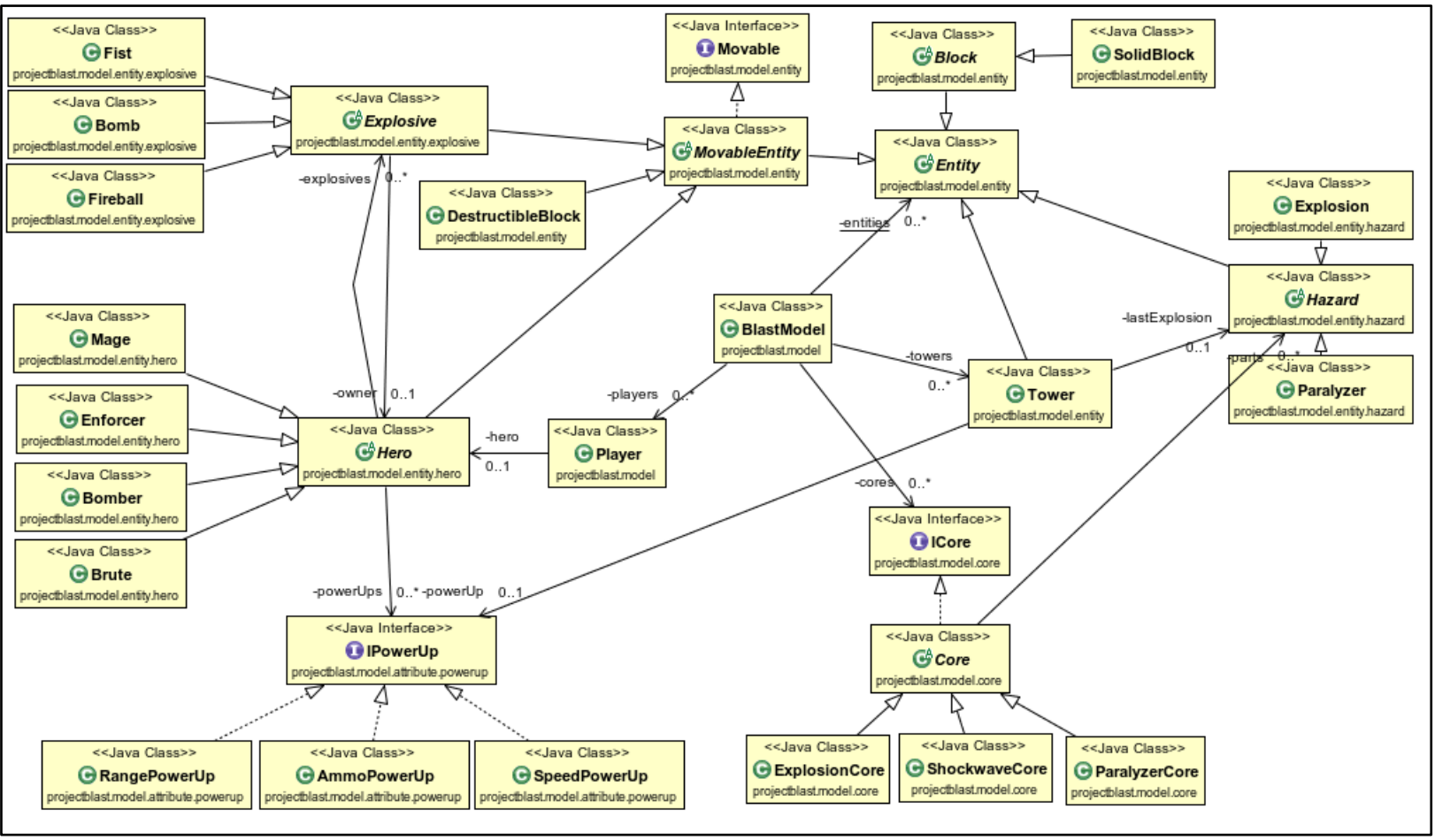

Figure 2. Class diagram for the Model package.

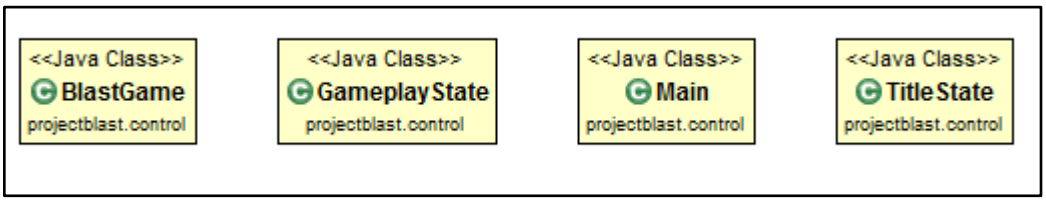

Figure 3. Class diagram for the Controller package.

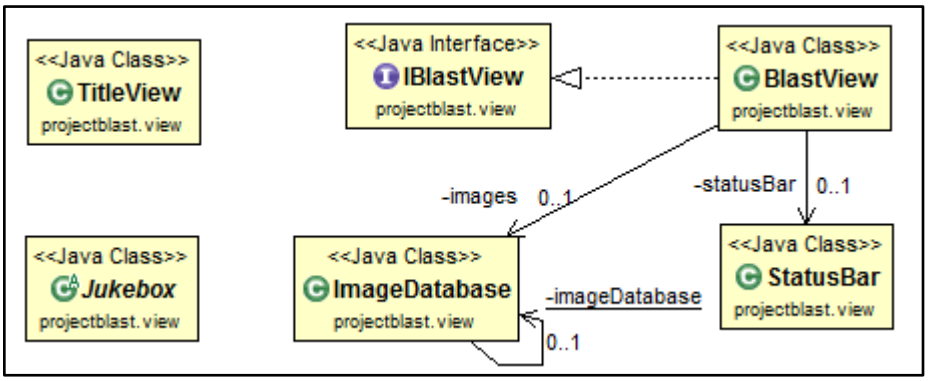

Figure 4. Class diagram for the View package-

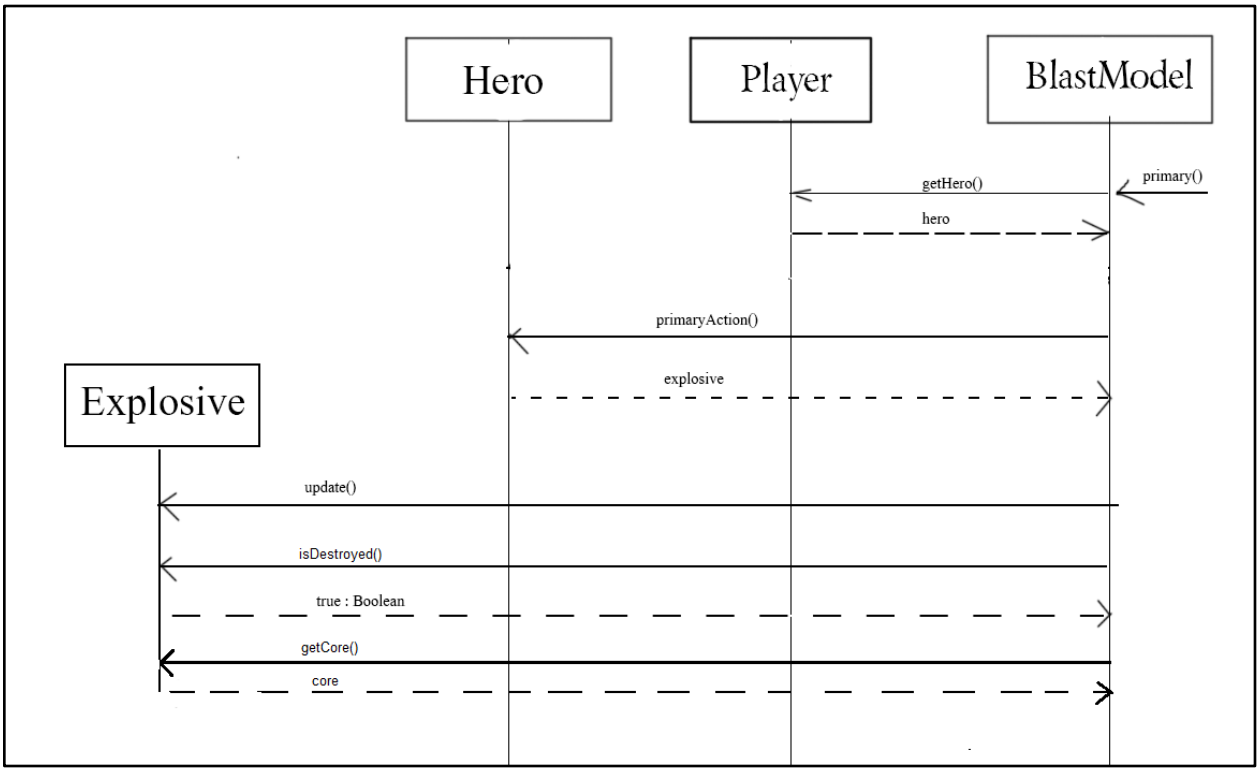

Figure 5. Sequence diagram for use case PrimaryAction

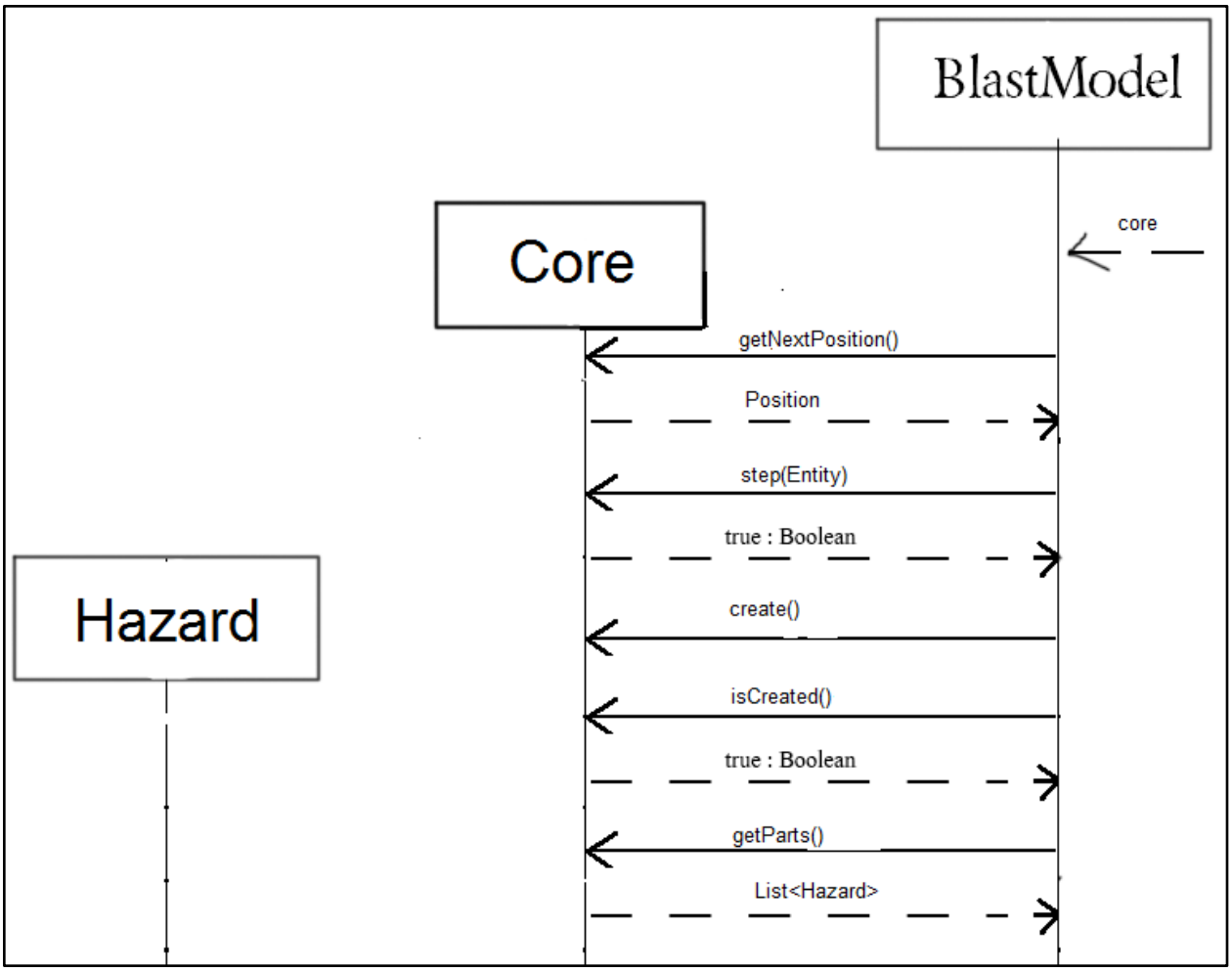

Figure 6. Sequence diagram for use case CreateBlast