System Design Document (SDD) for Blast in the Past

TDA367

Group 21:
Mattias Johansson
Bjarne Gelotte
Jonas Graul Sagdahl

Version: 1.0
Date: 2015-05-27
Authors: Mattias Johansson, Bjarne Gelotte, Jonas Graul Sagdahl

This version overrides all previous versions.

1. Introduction

1.1 Design goals

The project structure should be categorized in different packages following the MVC-model. The packages should be loosely coupled and have a strong coherence. If a package were to be deleted, the rest of the application should still work fine. Apart from the MVC-packages there will also be other packages such as utils and tests.

1.2 Definitions, acronyms and abbreviations

HP - Health Points, signifies the amount of damage a character can absorb before dying.
2D - Two Dimensions/Two-dimensional
AI - Artificial Intelligence
Ammo - Ammunition
Boss - A particularly tough and dangerous enemy
Power-up - An item which gives the PC certain positive abilities like increased damage or faster movement for example. Expires after a short time (less than a minute).
NPC - Non-Player Character, a character that is not being controlled by the user.
PC - Player Character
GUI - Graphical User Interface

2 System design

2.1 Overview

The application structure is based on the MVC-model. It also includes a package for utilities and a package for testclasses. The application uses the framework LibGDX, mainly for graphics and music. The Vector2-class from LibGDX is used for object movement and is the only part of the framework associated with the model.

2.1.1 The model functionality

The models functionality is divided into several packages. Each type of game object is contained within a package and implements an interface. BPModel handles the instansiation of game objects, aswell as updating them and eventually removing them.

2.1.2 Input handling

The input is handled first by InputHandler in the controller package. If the input is a keypress, InputHandler immediately sends the keycode forward to BPController.

BPController in it’s turn checks which controller is currently active and sends the keycode forward to that controller. The active controller tells the model and view what to do, depending on the input.
If the input is a mouse click, the chain of events is very similar. The InputHandler calls touchDown in BPController and BPController calls touchDown in the currently active controller.

2.1.3 Weapons

Weapons extend the abstract class Weapon. The Weapon class handles firing and reloading. When creating a concrete weapon it is necessary to extend the Weapon class, call the super constructor and send the correct parameters. It is also necessary to provide a concrete implementation of the getNewProjectile and getWeaponType methods. The getNewProjectile should return a new instance of a Projectile which matches the type of the weapon. This might require the creation of a new Projectile class.

2.1.4 Projectiles

Projectiles extend the abstract class Projectile.The Projectile class implements the ProjectileInterface. To create a new projectile, extend the Projectile class and call the superconstructor, sending the correct paramters. It is also necessary to implement the getProjectileType method.

2.1.5 Characters

There are two different types of Characters: Player and Enemies. All Characters have health, movementspeed and a weapon. The Player is controlled by the user, who can use the keyboard and mouse to move, aim and shoot. The Enemies contains of Pleb and Boss, where Boss is just a stronger version of Pleb. Enemies have a random movement pattern and spawns on a random location on the map. They can fire their weapons if the Player is nearby. Enemies can drop a random PowerUp when they are killed.

2.1.6 Ammunition

2.1.7 Power-ups

Power-ups are meant to enhance the player characters attributes such as movement speed etc. After an enemy is killed they have a chance to drop a power-up which the player character will be able to loot. Once it’s looted (simply by walking onto it) the power-up is activated for a limited time.

2.1.8 Factories

The project follows the Factory Pattern (see references) where the idea is to have a central class for creating instances of subclasses to a generic class such as Weapon.

2.1.9 Utilities

The utils package contains convenient classes that are used by all the other packages. These classes include:

- Constants - contains global constants
- GameData - contains high score data
- GraphicalAssets - contains variables for textures and images
- HighScoreHandler - handles java.io filestreams (for saving high scores)
- Position - used by objects in the game (Characters, Projectiles, etc.)
- PositionInterface
- Rectangle - used as a hit box for collision detection
- RectangleAdapter - adapts the Rectangle class in LibGDX to a Rectangle that the model can use
- SoundAssets - contains variables for sound and music

2.1.10 View, GameStates

In the view package lies all graphical representations of the model, including characters, projectiles and GUI. There are five different game states (or view states), MainMenu, PlayState, GameOverState, InGameMenu and HighScoreState. The active state is controller by GameStateManager.

2.2 Software decomposition

2.2.1 General

The application is decomposed into the following packages:

- model - contains the core with all logic. Model part of MVC.
- view - graphical representation of the model. View part of MVC.
- controller - handles input and delegate tasks to model and view. Controller part of MVC.
- utils - convenient classes used by other packages, e.g. Position, Constants, GraphicalAssets.
- tests - test classes mostly to test use case related classes’ functionality

2.2.2 Decomposition into subsystems

Model packages:

- ammunition - contains ammunition class and interface
- chest - contains chest class and interface
- enemy - contains enemy classes, i.e. Enemy, Boss, Pleb and EnemyFactory
- level - contains BPModel (which is the main model class), LevelInterface, LevelManager and level classes (that implement LevelInterface)
- player - contains the interface Character and the class Player
- powerup - contains PowerUpI (interface), PowerUp (abstract class) and different PowerUp-classes
- projectiles - contains ProjectileInterface, Projectile (abstract) and different Projectile-classes

- weapon - contains WeaponInterface, Weapon (abstract), WeaponFactory and different Weapon-classes

The interface Collidable lies directly in the model package. It is extended by Character, PowerUpI, ProjectileInterface and AmmunitionInterface.

View packages:

- characterviews - contains CharacterView (interface), CharacterViewFactory and different CharacterView-classes.
- gamestates - contains GameState (abstract), GameStateManager and different GameState-classes.
- projectileviews - contains ProjectileView (abstract), ProjectileViewFactory and different ProjectileView-classes.
- weaponviews - contains WeaponView (abstract), WeaponViewFactory and different WeaponView-classes.

The interface WorldObject, which is implemented by every entity class in the game world, can be found outside these packages along with AmmunitionView (The ammunition which enemies can drop when they are killed), ChestView, PowerUpView and PowerUpViewFactory.

2.2.3 Layering

The layering is visualized in the figures below. Packages higher up in a figure are at a higher level than the packages below.

2.2.4 Dependency analysis

As shown in Figures 1 and 2, there are no circular dependencies between packages.

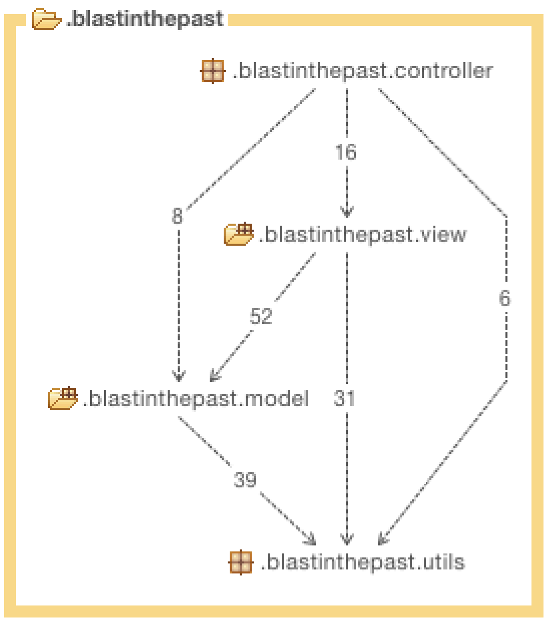

Figure 1. High level package structure and dependencies.

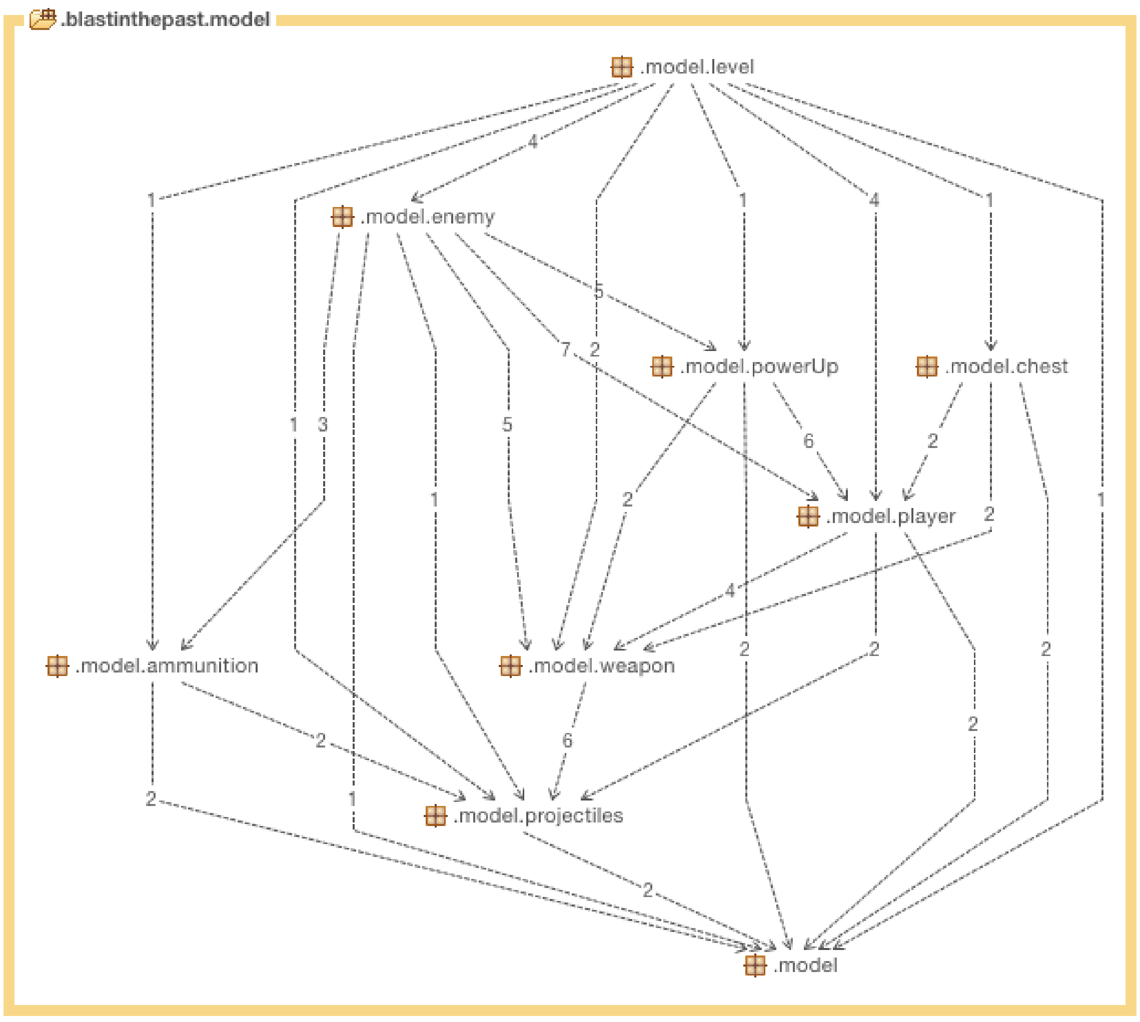

Figure 2. Level package structure and dependencies.

For further package structures and class structures within packages see Figure 3, Figure 4 and Figure 5 in appendix.

2.3 Concurrency issues

The program runs on a single thread so no concurrency issues.

2.4 Persistent data management

High scores are handled through filestreams using java.io in the HighScoreHandler class. On initialization, the program reads from a local .sav file containing the game high scores. If there's no such file, the program will create one with default values.

If a player manages to get enough points to get into the high score chart, the points and a three letter name will be written to the .sav file.

2.5 Access control and security

N/A

2.6 Boundary conditions

N/A. The application is launched and exited as a normal desktop application.

3 References

1. MVC, see https://msdn.microsoft.com/en-us/library/ff649643.aspx
2. Factory Pattern, see https://msdn.microsoft.com/en-us/library/ee817667.aspx
3. GUI, see http://en.wikipedia.org/wiki/Graphical_user_interface

APPENDIX

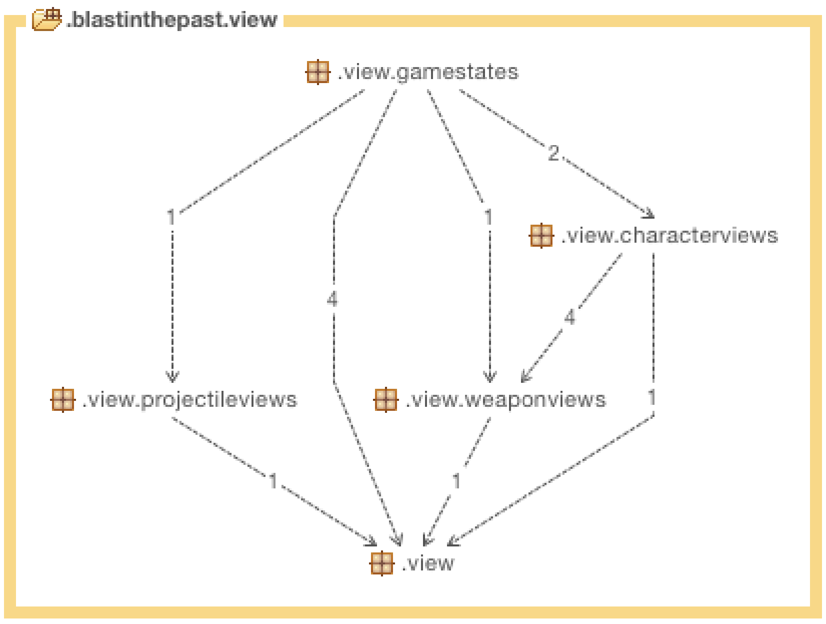

Figure 3. Gamestates package structure.

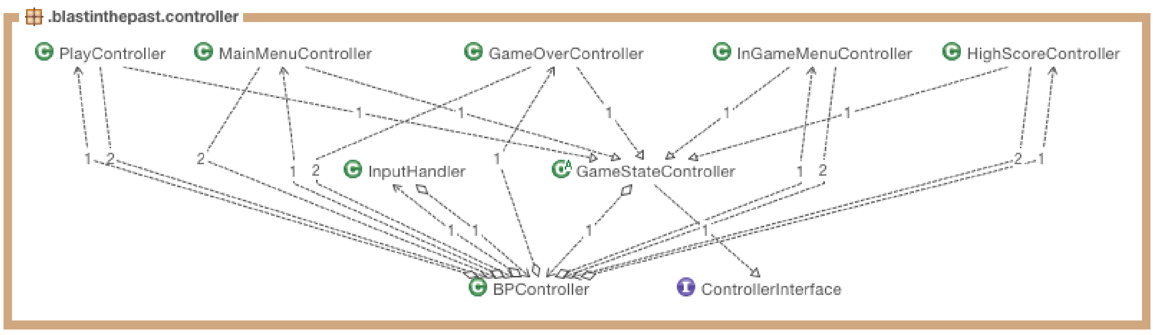

Figure 4. controller class dependencies.

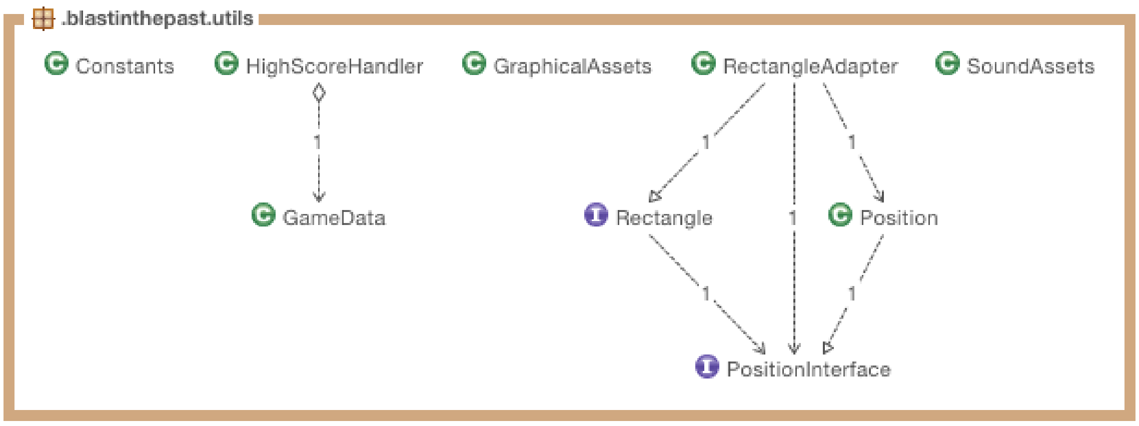

Figure 5. utils class depedencies.

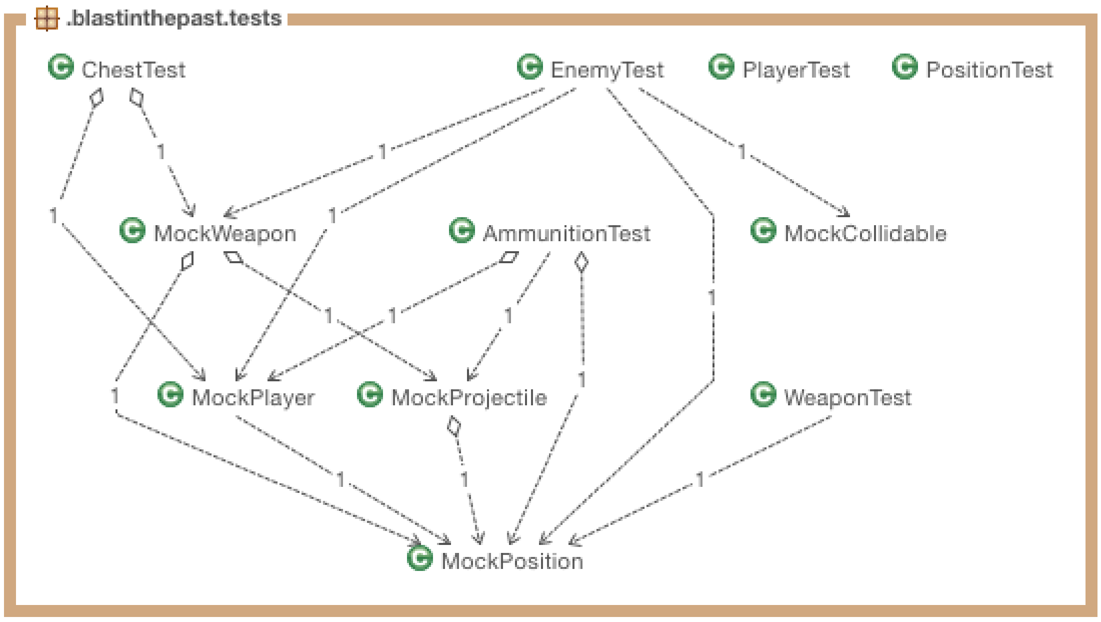

Figure 6. tests class dependencies