System Design Document for Ballbuster

Group 7

Table of Contents

Version: 2.0

Date: 2015-05-29

Author: Jacob Lundberg, Johan Segerlund, Joakim Eliasson, Matthias Andersson

1. Introduction

1.1. Design Goals

The design is to be based on images and sprites and give a modern impression of a 2D game. Meanwhile the design should be somewhat adaptable for the user (various skins, maps and colors).
The design might contain graphical content from other designers, but since we have no intention to release the game for the public this should not be a problem. The designs will be easy to replace for future modification.

1.2. Definitions, Acronyms and Abbreviations

All terms and definitions used within the BallBuster application are found below.

- GUI - Graphical user interface
- Java - Platform independent programming language
- JavaFX - A part of the java programming language, primarily used to build websites and applications with a similar form.
- JRE - Java Runtime Environment. The additional software required to run Java applications.
- MVC - Model-View-Controller. A code technique to avoid mixing of model and view code, with a middle part referred to as the controller.

- Ball - A ball contains a position, aura and a map. Each player will have one ball. The ball have information about the speed, which can be assigned. The ball also contains a shield which is represented by a double.
- Aura - Each aura contains one ball. You can set the aura status and also get the current aura state. Its also possible to set the position of the aura.
- Player - Player contains playerID, playerName and Ball. Player class is basically the player. It contains the assign keys for a player and the Ball.
- PowerUp - Power ups will spawn on the map and each player can take them by moving into them. The power up is activated as soon the player takes it. Each power up will have different abilities. PowerUp contains a string. The string will basically define what the powerUp will do. Each instance of powerUp will do different things to the players ball.

- Health Pack - Health Packs is like a power up but only have one ability which is to increase the life of the player who takes it. It will have a different sprite then the power up.
- Mana is required to be able to activate an abilities. At current state it gives the player a speed boost.

2. System Design

This chapter will describe how the application is implemented.

2.1. Overview

The project is built using a MVC design. Most of the models have their own controllers and views. All controllers except for the menu controller implement IController, which is an interface to replace the standard ApplicationListener.

2.1.1. Event Handling

The event handling is handled by InputProcessors and the IController interface. The InputProcessor in BallController will listen to keyboard input to move the ball on the playing field. Depending on which key is pushed, a method in the BallView is triggered.

For other events, such as collisions and gathering of powerups, controllers for the specific event will manage which methods are triggered. For example, if you hit a powerup tile with the ball, the PowerupController will check what kind of powerup the tile represented and then add the functionality to the player.

The application has a CollisionController that handles collisions between ball and walls.

As mentioned before, all controllers will implement the IController interface that contains the method onCreate() and onRender(), so it basically works as a ApplicationListener.

2.1.2. User Defined Variables

There are only a few user defined variables in the game. For example the user can decide which keys are used to control the ball. These have a default value, which means they can never be null and make the program crash. The user can also change map.

2.1.3. Saving and Loading

The application does not save anything therefore no loading occurs. The keybinds are always set to default.

2.2 Software decomposition

2.2.1. General

The application is split between a core- and a desktop module, built with an MVC structure.
The core module is the center of development.
Since the application is developed with MVC structure, the application holds model-, view- and controller packages.
Most of the models has their own controller- and view classes.
The model classes contains no graphical content, this is crucial to ensure full testability and make graphical library easier to replace.

2.2.2. Layering

The application is based on libGDX, and uses further implementations of the libGDX framework, such as scene2D and box2D.

2.2.3. Dependency Analysis

There are no misdirected dependencies. As figure 2 shows, the controllers interact with both models and views. Also, the views hold connections to the models. The test classes will only point to the models, which is a requirement.

2.3. Concurrency Issues

There are no more than one running thread in the application.

2.4. Persistent Data Management

The application does not save any data when the application is shut down.

2.5. Access Control and Security

N/A

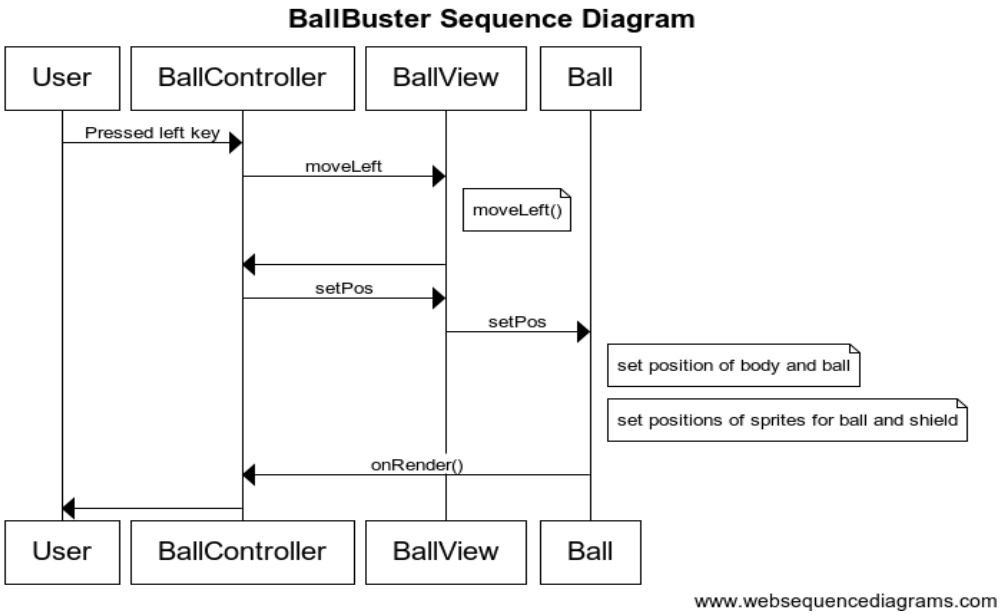
2.6. Boundary Conditions

N/A

3. References

N.A

Appendix



Figur 1: Sequence diagram for ballbuster

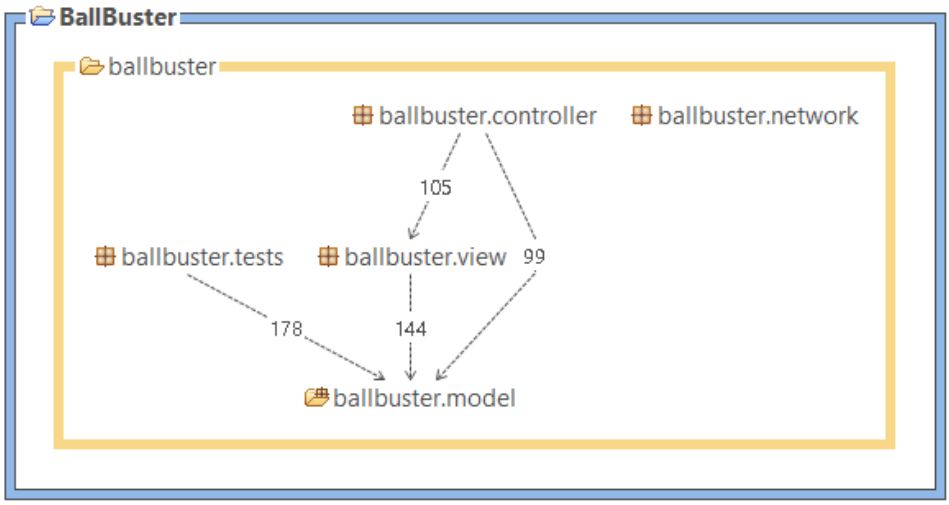


Figure 2: Dependency analysis for ballbuster

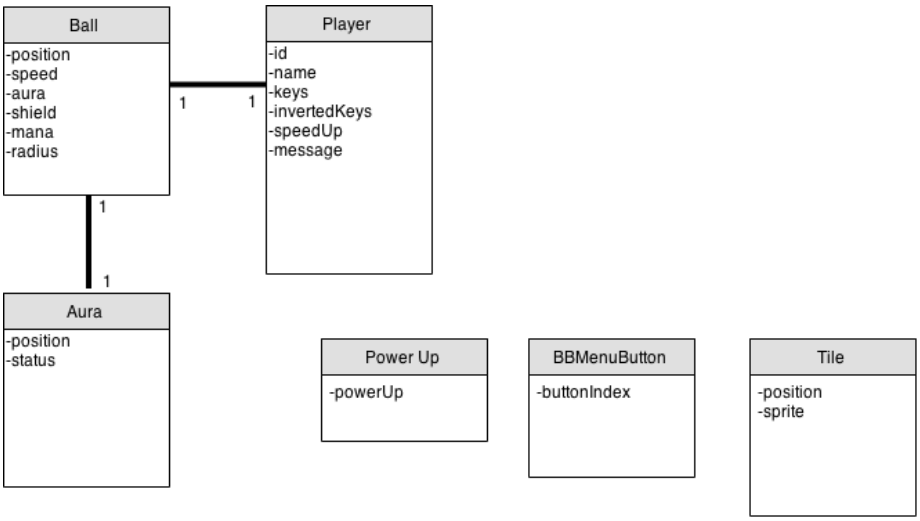


Figure 3: UML Domain analysis for BallBuster

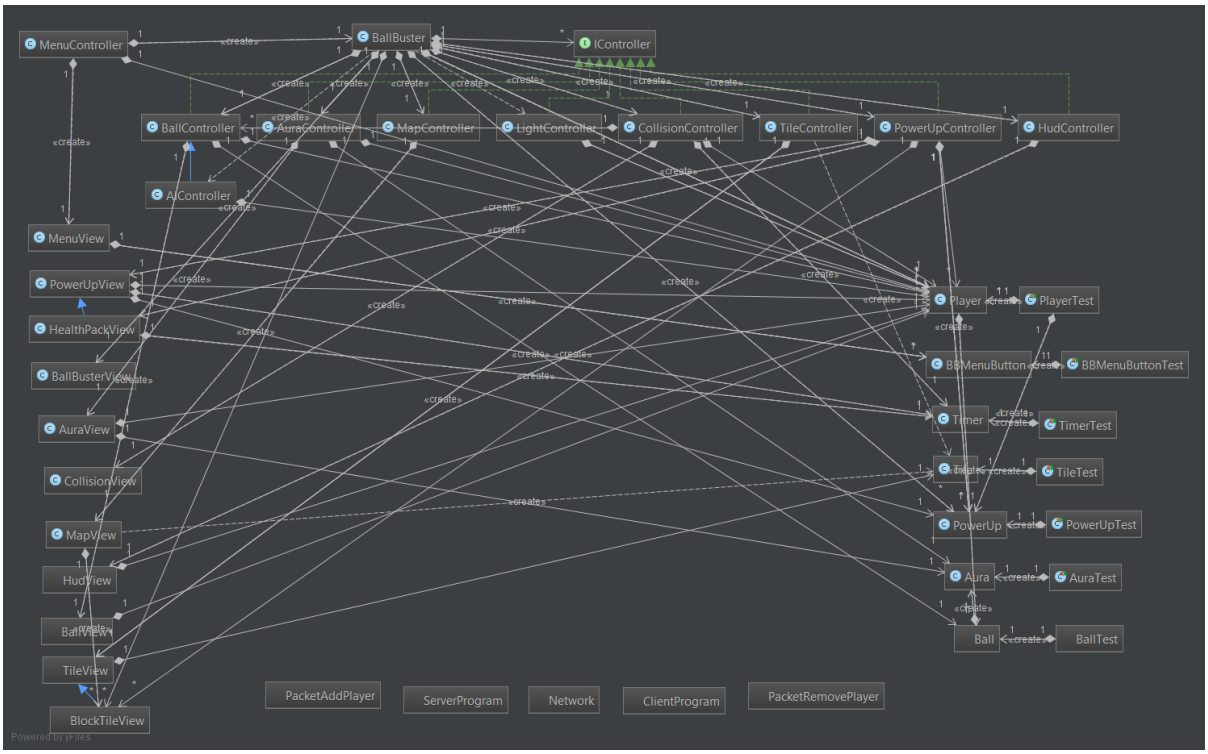


Figure 4: Complete UML for BallBuster, generated using IntelliJ