System design document for Blockster

Contents

Version: 1.0

Date 2014-05-25

Author Eric Bjuhr, Oskar Jönefors, Emilia Nilsson, Joel Tegman

This version overrides all previous versions.

1 Introduction

1.1 Design goals

The design will use the MVC design. The model will be independent from the view and controller classes, which will implement the libGDX framework. However, the model will be able to utilize other visual frameworks as long as the functionality for supplying the tilemap files is supported.

1.2 Definitions, acronyms and abbreviations

All definitions and terms regarding the core Blockster game are as defined in the references section.

- **GUI** - graphical user interface. Java - platform independent programming language.
- **JRE** - the Java Runtime Environment. Additional software needed to run a Java application. libGDX - The Game-development application framework that the application uses for the visual representations and user input in the game.
- **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 will use an MVC model. The model will handle internal representation of data for the players and the map. The visual representations for these will be handled by the view classes, utilizing the libGDX framework. LibGDX is also used for handling input in the Controller and for reading tilemaps of the tmx format and supplying them to the model upon initialization.

2.1.1 The model functionality

The game loop in the application resides in the libGDX render method. The method updates the controller which then sends it's commands to the model. After the controller update, the render loop sends an update command to the model, with the delta time between the last two rendered frames. The model then updates all it's objects and their movements accordingly. Finally, the View class is rendered on the screen.

2.1.2 Rules

The rules will be taken care of in the model classes, which will ensure that no invalid states can be set by the controller.

2.1.3 Unique identifiers, global hook-ups

We will not use any globally unique identifiers for any entity. The view will upon initialization create visual representations for all the players and blocks in the model, and utilize a map to connect a specific player or block to the generated view object.

2.1.4 Blocks and players

Blocks will make up the entire map and will be manipulated in different ways by the Players. These two share several methods for animation states, coordinates and scale that they inherit from the AbstractBlocksterObject class.

Blocks have different properties. The properties supported by the models are:

Solid - The player will collide with the block.
Movable - The block can be pushed or pulled.
Liftable - The block can be lifted and placed.
Weighted - The blocks will fall down if nothing solid is directly below them.
Portals - The goal in the game is to reach these portals. Once all players have done so, the game is won.

2.1.5 Event handling

Events such as input events will be taken care of in the Controller which extends the libGDX InputAdapter class to receive input. It will send the appropriate commands to the Model.

The MiniMap class is a BlockMapListener and listens to the BlockMap in the model for changes such as blocks that have been moved, and draws them on the screen.

The model itself is a GameEventListener, which listens to the players to see if they've entered any of the portals.

2.1.6 Internal representation of text

There is no text in the application, as a menu wasn't implemented.

2.2 Software decomposition

2.2.1 General

The application is divided into the following main packages:

- **core** - Contains the Model and a Factory interface which has to be implemented by the view. Also contains the interfaces for GameEventListener and the enum GameEvent.
  - **core.objects** - Contains classes to represent the world map, the players and the blocks.
    - **core.objects.interactions** - Handles the interaction states where the player is lifting or grabbing a block.
    - **core.objects.movement** - Handles the different movements of the blocks and the players.

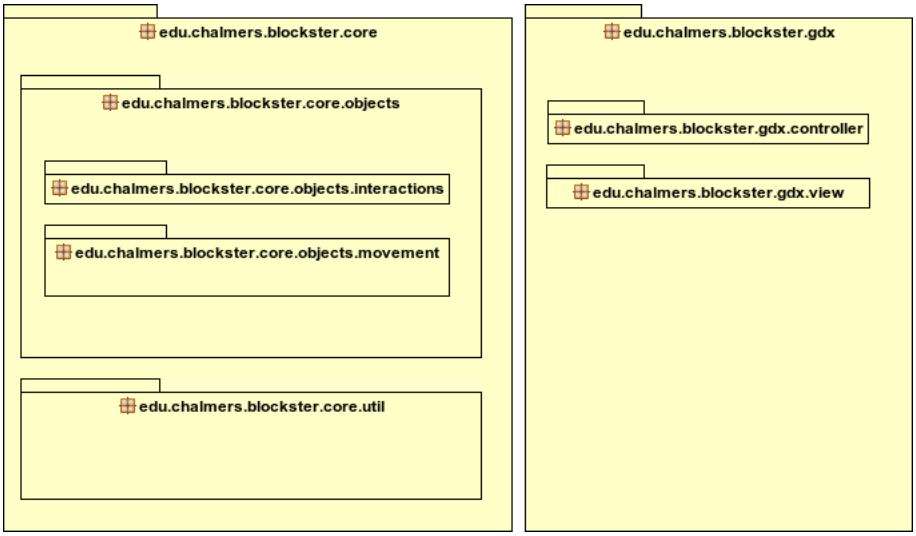

Figure 1: UML Package diagram showing the packages in the application.

  - **core.util** - Handles the calculations for collision detection, and the interfaces necessary to represent collidable entities.

- **gdx** - Contains the application launcher, and the Blockster class which runs the game loop. This package, and subpackages all contain libGDX dependencies.
  - **gdx.controller** - Contains the Controller which handles input from the user.
  - **gdx.view** - Contains the view classes and the factory classes for creating them.

2.2.2 Decomposition into subsystems

The GdxFactory class uses LibGDX's TMXMapLoader class to read the tmx files representing the game maps.

2.2.3 Layering

The layering is shown together with the dependency analysis below. To the left is the GDX package in relation to the core package, and to the right is the core package in collapsed form. Higher level classes are on top of the figure.

2.2.4 Dependency analysis

The internal dependencies are shown below. There are no circular dependencies.

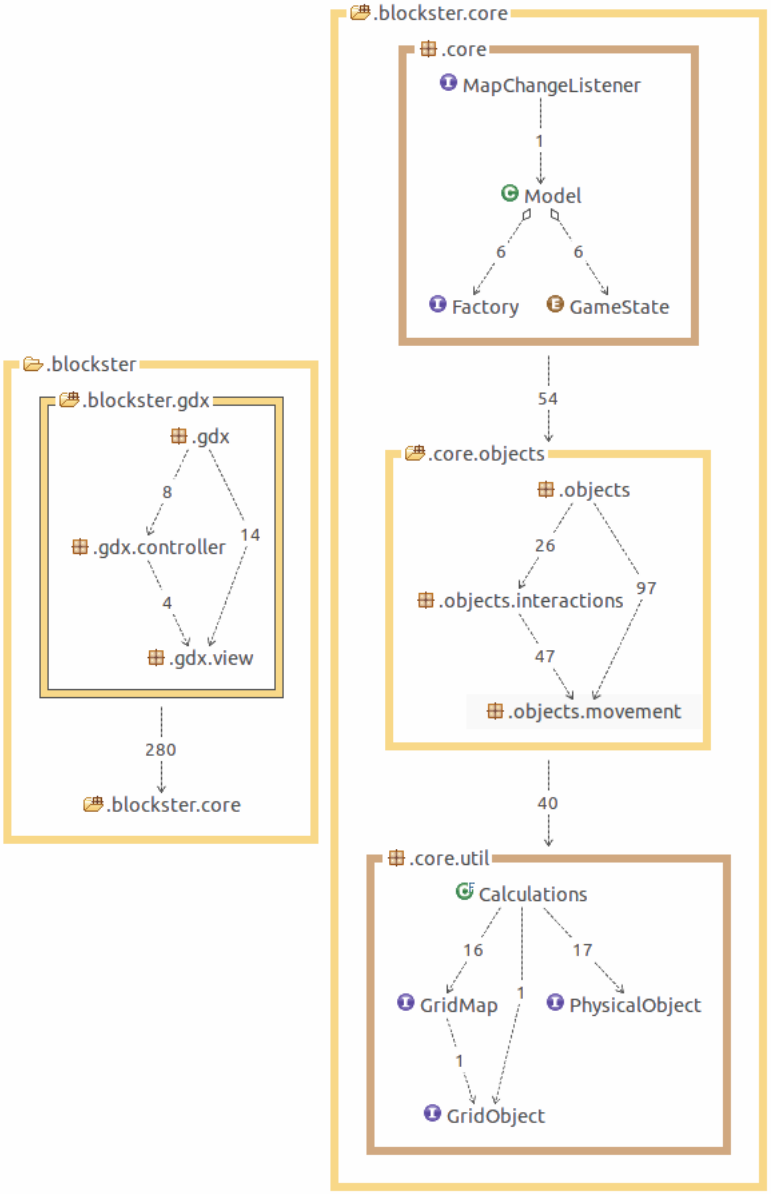

Figure 2: Figure showing the internal dependencies and layering of the application.

2.3 Concurrency issues

There is only one active thread in this application, so there are no concurrency issues in neither the model nor the view. However, the user input is event based and thus is taken into account when creating the controller. This is done by setting a volatile flag according to user input, and upon each rendering of the game we check the flag for user input and then change the model accordingly.

2.4 Persistent data management

No data is saved between sessions.

2.5 Access control and security

N/A - All users have the same access. There is no authentication mechanism or encryption.

2.6 Boundary conditions

N/A - Application is launched simply by running the executable jar file.

3 References

Block Dude: http://www.detachedsolutions.com/puzzpack/blockdude.php

LibGDX: http://libgdx.badlogicgames.com/

MVC: http://en.wikipedia.org/wiki/Model-View-Controller