The System Design Document for “Roung Out”

Version: 0.3
Date: 28/05/2017
Author: Alex Nitsche
Group: 30
Participants : Alex Nitsche, Adam Grandén & Ken Bäcklund

Table of contents

1. Introduction

1.1 Design Goals

This document describes the construction of the RoungOut game-application developed during this project, our design goals are the following:

- The model must be testable
- The project should use the MVC model

1.2 Definitions, acronyms and abbreviations

All definitions, acronyms and abbreviations for the project and for this document can be found in the RAD or references section as a document or definition.

2. System Architecture

Description

The application uses the libgdx framework and it's designed to be run as a desktop application on one device. We have made our application with the windows 10 operating system in mind and as the primary platform for operations.
The application uses a MVC model. The model is the unit that handles the core game, the controller's responsibility is to handle inputs from a user to the model and view, the view is outputting the representation of the model to the user. The layout of the application can be seen in *Figure 1* in the dependency analysis section.

2.1 Dependency Analysis

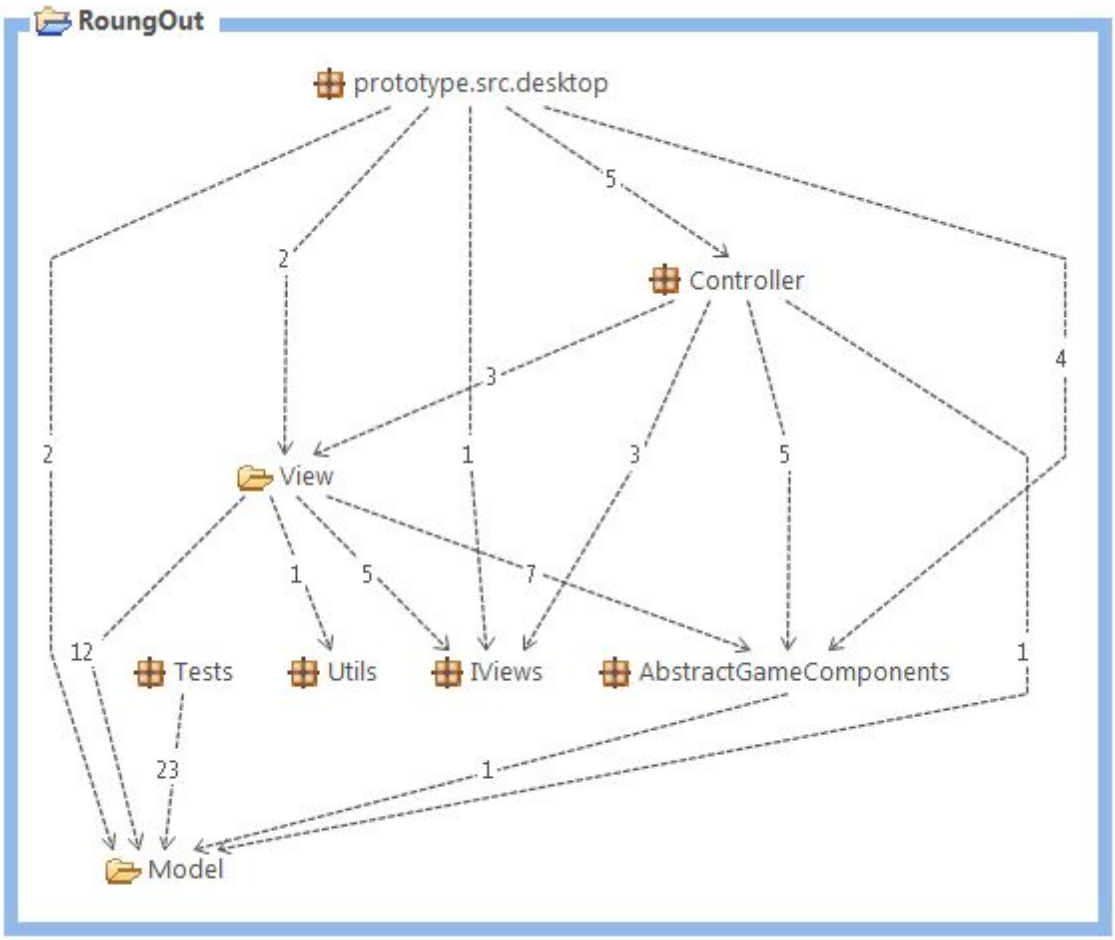

Figure 1

prototype.src.desktop

Where the application starts there when it's launched, it does some general initialization of the application.

AbstractGameComponents

Contains abstract classes that is used to remove circular dependencies between prototype.src.desktop and the view and the controllers.

View and IViews

The top level package for all view related things, for example the main screen and the option menu, the views uses the libgdx framework. IViews is an Interface that connects the Controller to the view

Utils

Contains a self made tool to ease the creation of visualisations of parts of the model.

Controller

Handles inputs from the user via the use of the libgdx framework

Model

The model contains the game itself. It contains a physics sub-system that define the shapes of rectangular and circular object. It also contains a collision sub-system that uses the bodies the physics system creates. The Collision subsystem is responsible for handling all of the collisions between 2 bodies. These subsystems are self made and not imported from a library or framework.

Tests

Contains all the tests, it uses junit5 for the tests.

Dependencies

As we can see in *Figure 1* there is no circular dependencies in our project, although prototype.src.desktop is depending on almost everything it's because it does the initialization.

2.2 General observations

Initialization

The model,view and controller are initialized at the class RoungOut located in prototype.src.desktop. The class extends "AGame" which is an abstract class whose sole purpose is to remove circular dependencies from the project. AGame extends "Game", a class in the libgdx framework, its purpose is to ease handling of "Screens" which is used in the views.

Controller Handling

The Controller packages has a handler and an interface for the handler and also a set of enums. The handler's purpose is to delegate the right to have the input. In libgdx only one InputProcessor can be active at any given time, the active inputprocessor gets called on a number of keyboard inputs. The handler has a list of the controllers and each controller has a instance of the handler, if a controller wants to give over the input to another controller the handler is used together with a enum, the enum corresponds to the receiving controller.

Interfaces

A majority of the objects that is created in the application are instances of various interfaces. These interfaces results in very few (if any) circular dependencies inside of the packages. They also provide a general and logical structure for the inner workings of the packages.

The game loop

The game loop is the loop in which manny things updates the frames are rendered, the pads move one-unit, the ball moves and updates a positions. The loop is like the clock of the game and it is currently implemented with the help of the framework.
The screen that currently is active is the one that gets called by an update method in start of each frame. When the someone is playing the game, the active screen is a View (BoardView) that updates the model and then updates the views. We break the norm of updating the model in the controller and instead updates it in the view because of the screen handling the framework uses.

3. Subsystem decomposition

3.1 Physics

This is the subsystem that is responsible for creating all of the bodies the collision system uses. The subsystem is located inside the Model package. The system can create two things, either a body with rectangular shape or a body with a circular shape. These bodies have certain data which is stored as a Location object. A Location in our application refers to the position, the direction, the speed and the max speed of a point in the 2D-plane. The Location object can also do some simple arithmetic operations (the most complicated being the pythagorean theorem) which aids and simplifies the handling of bodies.

3.2 Collision

Responsible for handling all the collisions, it uses the position of an object and its sizes, if it's a circular shape it uses the radia, if it is a rectangular shape it uses the shapes width and height to calculate the distance between two objects. And if that distance is close enough the objects each receive the collision call.

These calls contains the objects that it collided with, so if a circle hits a rectangle the circle gets a call with the rectangle object as a parameter and the rectangle gets the circle object as the parameter. Doing it this way allows object to do different things depending on what they collided with, we are currently not using this method but the possibility remains.

3.3 Tools (Vector)

Located in the Utils package, there exists a Vector class. It was made in order to help with the mathematics that is needed to tilt an object around a centerpoint. The subsystem serves as a tool in a toolbox and it is modular so it can be used in other instances as well.

4. Persistent data management

No data is collected during the time the application is running. The application has some amount of stored data (apart from the code itself), the data is in form of .png and .xcf and it's pictures that the applications uses during run-time. The data is stored in a package called "Assets" at the top level and is used by the View package to display things such as the splash screen.

5. Access control and security

N/A this project is a local desktop application, thus no access control and security features are necessary for the application.

6. References

Libgdx Website
Libgdx Documentation
RAD for the project can be found In the documentation Folder in the git-repository
WordBook can be found in the top-level Folder in the git-repository