System design document for Twirl

Version: 0.1

Date 19/4 - 2013

Author Eric Arnebäck, Linnéa Andersson, Andreas Wahlström, Johan Gerdin

This version overrides all previous versions.

Table of contens

1 Introduction

1.1 Design goals

The design should use MVC in a way that makes it easy to reuse the model when porting the game to other platforms. The physics engine should also be easily replacable. The model should be testable.

1.2 Definitions, acronyms and abbreviations

(TODO: write definitions of terms here that we use throughout the document)
Gramlich - AndEngine i fågelperspektiv.

Det var en afton i början av maj. Den lilla trädgården på Mosebacke bredvid servarna för AndEngine-forumen hade ännu icke blivit öppnad för allmänheten och rabatterna voro ej uppgrävda, istället satt i rabatterna Nicolas Gramlich stupfull; snödropparne hade arbetat sig upp genom fjolårets lövsamlingar och höllo just på att sluta sin korta verksamhet för att lämna plats åt Gramlichs svällande ego, vilken just tagit skydd under ett ofruktsamt päronträd; syrenerna väntade på sydlig vind för att få gå i blom för att sedan bli nedtrappande av Gramlich i sitt berusade tillstånd, men lindarne bjödo ännu kärleksfilter i sina obrustna knoppar åt bofinkarne, som börjat bygga sina lavklädda bon mellan stam och gren för att fly Gramlichs arme på 20tjog apor med tillhörande skrivmaskiner, vilken även var ansvarig för en majoritet av AndEngines källkod; ännu hade ingen mänskofot trampat sandgångarne sedan sista vinterns snö gått bort och därför levdes ett obesvärat liv därinne av både djur och blommor, ända tills Gramlich i sitt berusade tillstånd nådalöst trampade över dem, på samma sätt som han trampat på ninst 40dussin programerares hopp och drömmar genom sin totala avsaknad av förståelse för parellellprogrammering. Gråsparvarne höllo på att samla upp skräp från AndEngines alla krasher, som de sedan gömde under takpannorna på navigationsskolans hus; de drogos om spillror av rakethylsor från sista höstfyrverkeriet organiserad i ära till Gramlich, de plockade halmen från unga träd som året förut sluppit ur skolan i

parellellprogrammering - och allting sågo de! De hittade elaka teckningar på Gramlichs miniskula könsorgan och kunde mellan stickorna på en bänkfot draga fram hårtappar efter Gramlichs aparme, som icke slagits där sedan Den Stora Refaktoriseringen i fjor, där otaliga programmerare hade stupat. Där var ett liv och ett kiv, och det blev ännu värre när Gramlich själv kom där gående, stupfull som han var.

2 System design

2.1 Overview

The game uses a MVC model.
(TOOD: describe the model)

2.1.1 Entities

All the ancestor of all the entities is the abstract Entity class. But no entity used in the game directly extends Entity, they rather extends one the classes CircleEntity, PolygonEntity, or RectangleEntity. Which one they chose to extend depends on their shapes. I.e., Obstacles are polygon-formed so they extend PolygonEntity.

2.1.2 GameWorld

GameWorld implements IGameWorld, which is the top-level interface exposing functionality of the model.

2.1.3 Physics

In the physics package there are a few interfaces meant to be implemented by whatever physics engine thats going to be used by the model. There is also a interface for the abstract Entity class so that all entities could be safetly used by the physics, this interface only have the methods needed by physics.

2.1.4 Level generation

The application implements an RandomLevelGenerator wich is responsible for creating all entities and placing them at empty positions, it is also here level size is decided based on how far the game has progressed. (TODO on sat).

2.2 Software decomposition

2.2.1 General

- controller, the controller classes.
- model, the entities of the app and the gameworld. There are also some package private classes in there for managing entities.
- model.powerup, the powerups of the game.
- model.resource, basically classes for mapping resources to file names. These resources are then loaded by displayed by the view. The classes for managing the high score list and localization strings are also here.
- model.physics, interfaces that describe the physics functionality that the model needs. These interfaces are then implemented by the view(TODO; mention somewhere in the beginning that the view handles the physics). There is also a vector class here that we nicked from box2d
- model.randomlevelgenerator, classes used for generating new, random levels in the game. The main class here is the RandomLevelGenerator, which is used by the model to generate the random levels.
- model.randomlevel.generator.sat, In randomly generated levels, we must ensure that entities are not placed on top of each other, and so we need some kind of collision handling. Box2D was to inflexible for this purpose, so we implemented collision handling from scratch using the Separating Axis Theorem. That theorem is basically implemented by this package.
- View, the view releated classes.
- View.scene, Scenes are separate screens of viewing in AndEngine, for example, the high score list is one scene and the main menu is another scene. All the scenes of the game are stored here.
- View.entity, the views of all the entities are stored here.
- View.loading, utility classes for the loading scenes are stored here.
- View.andengine.entity, AndEngine also has a class called Entity. We needed to extend this Entity clss a couple of times for some classes in the view, and these classes are stored here.
- View.resource, the resources that the model wishes to show to the user(sounds effects and images) are stored here, and of course shown by the view. As familiar,

the paths to these resources can be found in the model.resource package.

- View.loader, classes that simplify the loading of resources in AndEngine
- view.opengl, AndEngine has no support for primitives like polygons and circles, so we had to implement that from scratch, using opengl. The classes that handle this are put in this package. Also, a class that draws a star background using opengl is also stored here.
- View.physics, these classes expose the physics functionality from Box2D that we need. The interfaces in model.physics are also implemented here.

Package diagram. For each package an UML class diagram in
appendix

2.2.2 Decomposition into subsystems

We don't have any subsystems. Everything is handled in one, big system.

2.2.3 Layering

The layering is as indicated in the Figure below . Higher layers are at the top of the figure.

2.2.4 Dependency analysis

Dependencies are as shown in Figure. There are no circular dependencies.
(TODO: insert a Stan generated diagram below to show the laying and dependencies.)

2.3 Concurrency issues

Practically all multithreading is handled by AndEngine. The only times where we use our own threads are during the loading scenes and when removing all bodies from the physicsworld. The loadings scenes utilizes a worker thread to load all game resources, meantime in the main thread a loading animation is shown. However, those are shown for a such a short period of time, plus they don't listen to any user input, so they should not pose any problems. The reason there is a separate thread for removing bodies is because there is a risk of the physics engine crashing if not done this why.(TODO more on this new thread)

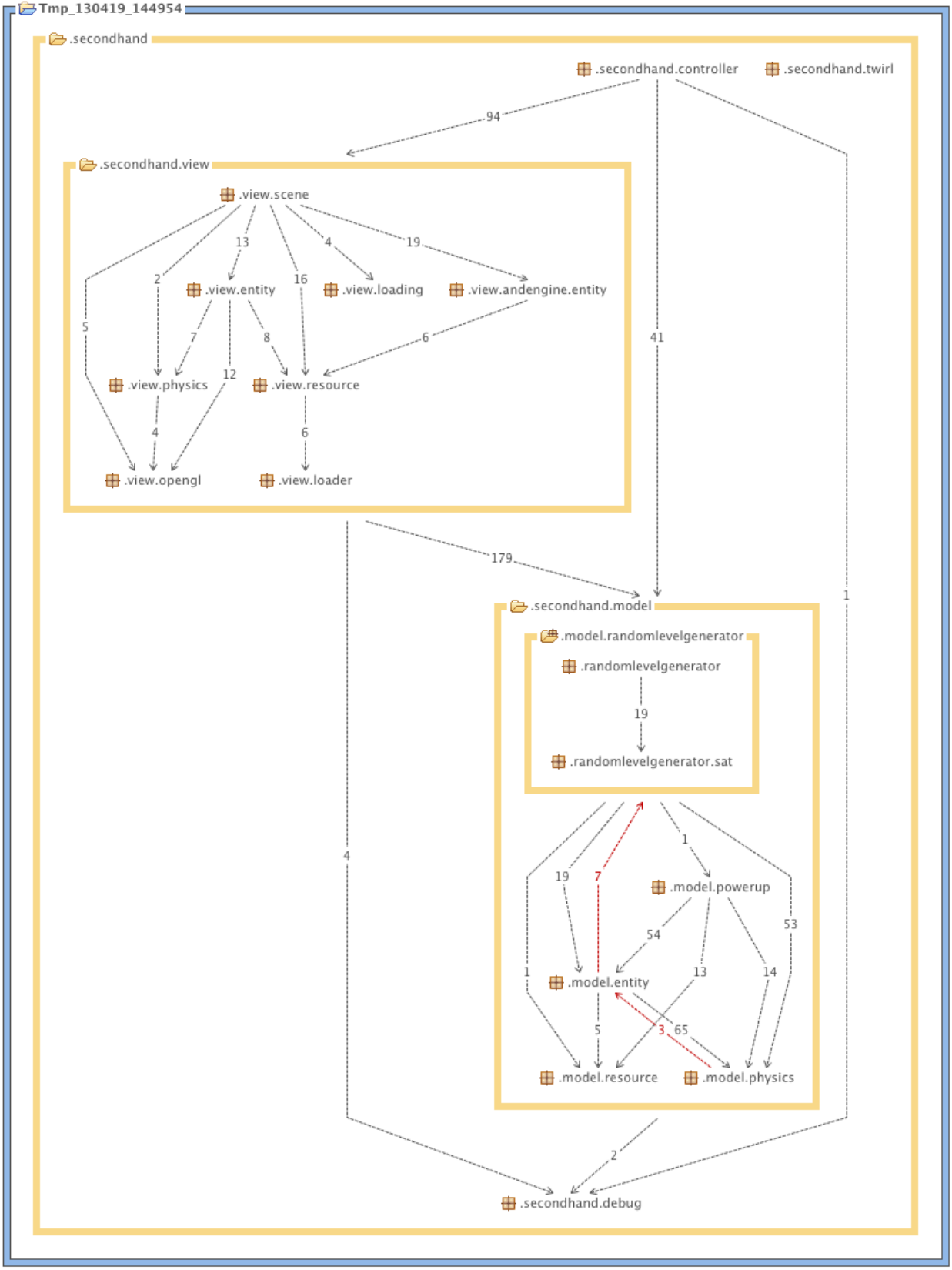

Figure: Package structure

2.4 Persistent data management

- Localization files for the localizible text of the app will be stored using XML, using the system for localization offered by the android OS, using strings.xml files. See http://developer.android.com/guide/topics/resources/localization.html
- The high score list is stored using a flat ASCII-based format. See the appendix

2.5 Access control and security

NA.

2.6 Boundary conditions

The app is launched like any other android app, by clicking on the icon. It is exited by pressing the back button. By pressing on the home button the application will freeze and wait for the user to procede playing when he/she reenters the app.

3 References

(TODO: Cite important design sources here. Ie, mvc. Not anything related to code details!)

APPENDIX

Class diagrams for packages

(insert class diagram of the mode.entity package here)

- Entity, the base class that all entities extend from
- CircleEntity, circle-formed entities derive from this class
- PolygonEntity,polygon-formed entities derive from this class
- RectangleEntity,rectangle-formed entities derive from this class
- BlackHole, all types of black holes(enemy and player) derive from this class. Has functionality for eating other entities.
- Enemy, an enemy, AI-controlled, black hole
- Player, a player-controlled black hole.
- Obstacle, an uneatable, polygon-formed, obstacle in the game,
- Planet, an eatable planet in the game.
- IPowerUp, interface that all powerups classes implements.
- CollisionResolver, helper-class resonsible for resolving potential collisions between

entities.

- EntityManager, manages all the entities in a list, and is responble for updating them every frame, and makes sure that removed entities are removed from the world.
- GameWorld, the world in which the game takes place. Responble for transitioning to the next level. Uses RandomLevelGenerator to create the next level, and EntityManager for managing all the entities of the world.

File Formats

The format for storing the high score list is a simple ASCII-format simple. A list of 5 persons is stored as follows:

Name
score
Name
score
Name
score
...

So name and score is separated by a newline character. And the separate entries are too separated by a newline character The file is assumed to be sorted, so the first name found in the file is the one with the highest score.