System Design Document for NeedForSpeed (SDD)

Contents

Version: 1.0
Date: May 28, 2015
Authors: Anton
This version overrides all previous versions.

1 Introduction

1.1 Desgin goals

The design must be modular to have to possibility to switch GUI ..

1.2 Definitions, acronyms and abbreviations

- GUI, Graphical User Interface
- Java, platform independent programming language.
- MVC, a way to partition an application with a GUI into distinct parts avoiding mixing GUI-code and application code.
- JSON, fileformat used for transmitting data in a structured way.
- Android, mobile operating system.
- libGDX, framework for developing games in Java.

2 System Design

2.1 Overview

The application uses a modified version of the MVC pattern for Android.

2.1.1 Model Functionality

The entry point to the model is the World class. We split the model into diffrent classes such as Player, Enemy etc.

2.1.2 Global lookups

We keep track of the game state with a global variable.

2.1.3 Event handling

Event handling for the game is put in the PlayerController class. Event handling for menus and back-key is handled by their respective Screen class.

2.2 Software decomposition

2.2.1 General

INSERT PACKAGE DIAGRAM etc..
The application is decomposed into two main packages, android and core, this is required by libGDX. the android package is for android specific code and in the core package the application code.

The core package is futher decomposed into:

- view, the GUI part of the application.
- screen, the GUI part of the application.
- controller, the controller classes for MVC.
- parallax, for parallax background.
- highscore, highscore module for reading / writing highscore from file.
- gamestate, keeps track of the game state.
- model, game logic for the application, model part of MVC.

There is also a package for testing (should this be here??)

2.2.2 Decomposition into subsystem

The only subsystem present are parallax and highscore.

2.2.3 Layering

see figure 1

2.2.4 Dependency analysis

see figure 1

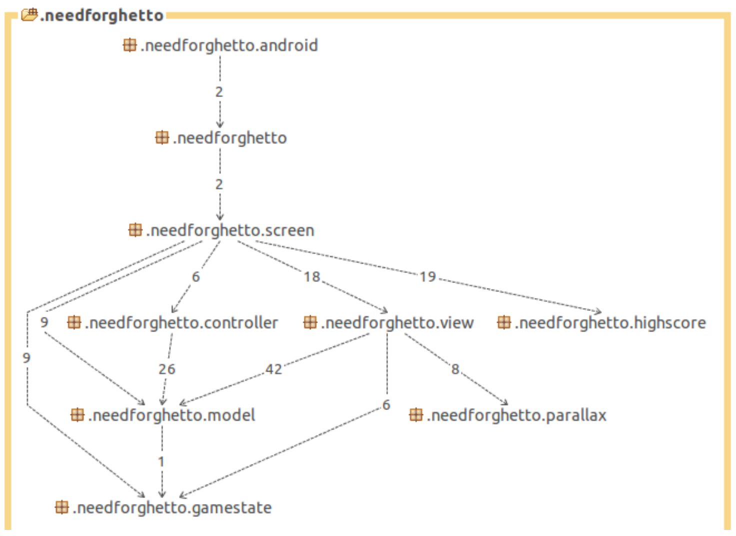

Figure 1: Layering and Dependency analysis

2.3 Concurrency issues

All concurrency is abstracted by libGDX. But interally libGDX renders on a separate thread(is this needed? is this even true, probably...).

2.4 Persistent data management

Assets (add to definitions ?) ,such as textures, are persistent and operated on with functions provided by libGDX. Highscore data is handled through the preferences (add to definitions?) interface.

Level design data is stored as JSON files.

2.5 Access control and security

NA

2.6 Boundary conditions

The application is launched and exited as a normal android application.

3 References

1. ref1
2. ref2
3. ...

Appendix A Class diagrams

Appendix B Other stuff