# 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.
- asset, binary file that contains e.g. sound, texture or font data.
- preferences, interface used to write persistent data.

# 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, to keep a modular design.

### 2.1.2 Global lookups

The state of the game is tracked with a global variable. The game state can take on the values paused or running, and is used to decide if we want to render or not in the gamescreen class.

There is a special state called godmode which is mainly used for testing purposes, it basically makes the player invincible.

### 2.1.3 Event handling

Generally when the application gets an input from the user it is handled by the controller package which then updates the model accordingly. The view is continuously rendered so a callback from the model is not needed.

Event not handled this way is buttons in menus and the back-key, they are handled directly in the corresponding Screen class. The motivation for this design choice is that these input have nothing to do with the model.

## 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 assets. The application code resides in the core package.

The core package is futher decomposed into:

- view, the GUI part of the game screen, more specific the rendering.
- 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.

The rendering of the game is decoupled from the GameScreen class to make it easier to debug, modify and add code.

There is also a package for testing.

### 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

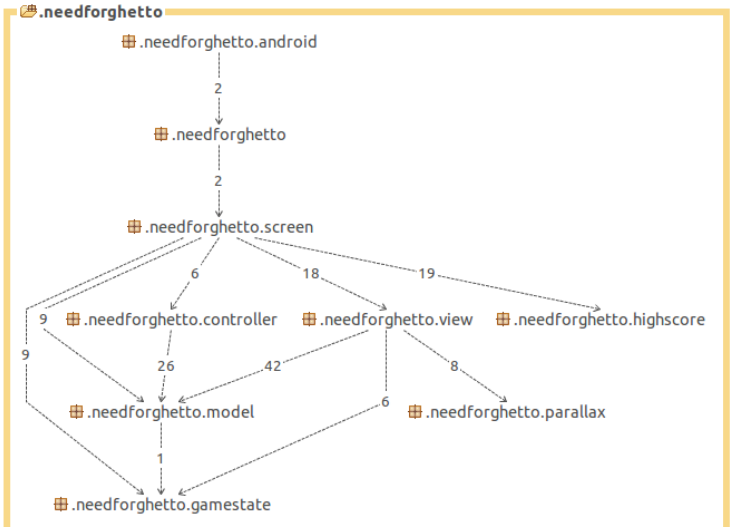


Figure 1: Layering and Dependency analysis

### 2.3 Concurrency issues

All concurrency is abstracted by libGDX.

### 2.4 Persistent data management

Assets, such as textures, fonts and level information, are persistent and operated on with functions provided by libGDX. Highscore data is handled through the preferences 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