System Design Document for Beat to the Beat

Version: 2.0
Date : 2014-08-21
Author : Malin Thelin
Revised by: Hampus Dahlin & Pontus Eriksson

This version overrides all previous versions.

1 Introduction

1.1 Design goals

The project's main goal is to write a program that has many possibilities for further development. We want the project to work simply, and to make changes or add further classes and methods should work seamlessly.

Especially our goal is to make the project work according to the "Open/closed principle". Our game is going to be designed to be a base for which one could further advance development or make a different game with our source code.

Crucial to our design is the music part, where we want to find beats in a song and spawn enemies, or whatever your imagination wants, in time with these. From this source code from our application one could design a vast number of different games.

1.2 Definitions, acronyms and abbreviations

Player: A person who plays the game.
Open/Closed principle: In object oriented design the principle states "software entities should be open for extension but closed for modification".
MVC: In object oriented design MVC means "Model-View-Controller" and is a way to program an application where the user sees the View, which is a visual representation of the Model, which in some cases is controlled by or uses the Controller.

2 System design

2.1 Overview

The application will use a modified MVC model, as required. The Model represents the game from a "logical" aspect, while the View visually represents the Model.

2.1.1 The controller functionality

We have a logical representation for each of the elements in the game and one controller for each of the major aspects of the operations needed.
HeadControl has an important role and this is the class that will delegate work to the other controllers. In HeadControl a method called startGame() will be placed which will, when used, start a timer that constantly checks for updates and tells the other parts of the program what to do next.

2.1.2 The model functionality

The different models: Actor, Environment, Music and Powerup are divided into larger packages since they all require subclasses and “help”classes et cetera. Actor for example is extended by two subclasses NPC and PC in order to inherit functionality.

2.2 Software decomposition

2.2.1 General

The application is divided into the following modules:

- *controller*, the different controllers. Control parts for MVC.
- *gui*, the visual representation of the model. View parts for MVC.
- *model*, the logical representation of the game.
- *model.actors*, the different characters that will be seen on screen. Both enemies and the player. Model parts for MVC.
- *model.environment*, model classes for how the environment will work and to build an environment. Model parts for MVC.
- *model.music*, model classes for the musicplayer(s) and analyzer. Model parts for MVC.
- *services*, holds FileHandler and other helpclasses.

2.2.2 Decomposition into subsystems

The only subsystem used is Minim which is used to to detect the beats.

2.2.3 Layering

At the top we have the controllers which affect the gui and model, controlling the whole flow of the program.

2.2.4 Dependency Analysis

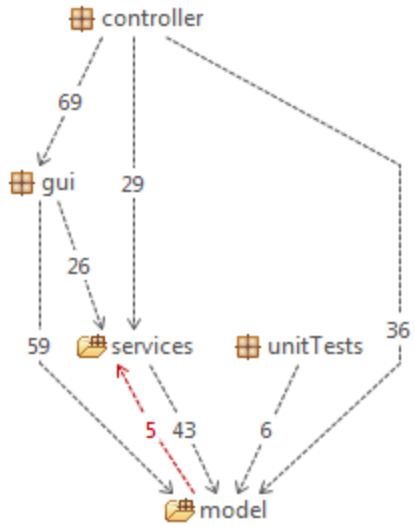

Figure.1 High level design

Songs have a HighScoreList, which is located in the services package, hence the coupling.

2.3 Concurrency issues

None.

2.4 Persistent data management

Options and songlists will be saved between runtimes.

2.5 Access control and security

N/A

2.6 Boundary conditions

N/A, application will launch and exit as normal desktop application.

3 References

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

APPENDIX

File formats:
Options file saved as a .conf
Song list file saved as a .list