System design document for Beat to the Beat

Version: 0.0.1
Date : 2014-05-01
Author : Malin Thelin

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 controlled by the Controller who knows what to tell the View from the Model.

2 System design

2.1 Overview

The application will use a modified MVC model, as required. We have decided to let the controllers take a slightly larger role.

2.1.1 The controller functionality

We have several different controllers for each area needed to be controlled, with a HeadControl to bind them all together. 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 controllers what to do next.

2.1.2 The model functionality

The different models: Actor, Environment, Musicplayer and Levels 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. Furthermore

2.1.3

2.2 Software decomposition

2.2.1 General

The application is divided into following modules, see figure(insert number here).

- *actors*, the different characters that will be seen on screen. Both enemies and the player. Model parts for MVC.
- *environment*, model classes for how the environment will work and to build an environment. Model parts for MVC
- *controller*, the different controllers. Control parts for MVC.
- *musichandler*, model classes for the musicplayer(s) and analyzer. Model parts for MVC.
- *levels*, takes the environment, music and other factors and puts it together into a level.
- *main*, where the Main class will be
- *gui*, holds different panels made for the menu, resultpanel etc.
- *services*, holds FileHandler and other helpclasses.
- *support*, all the exceptions.

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

2.2.4 Dependency analysis

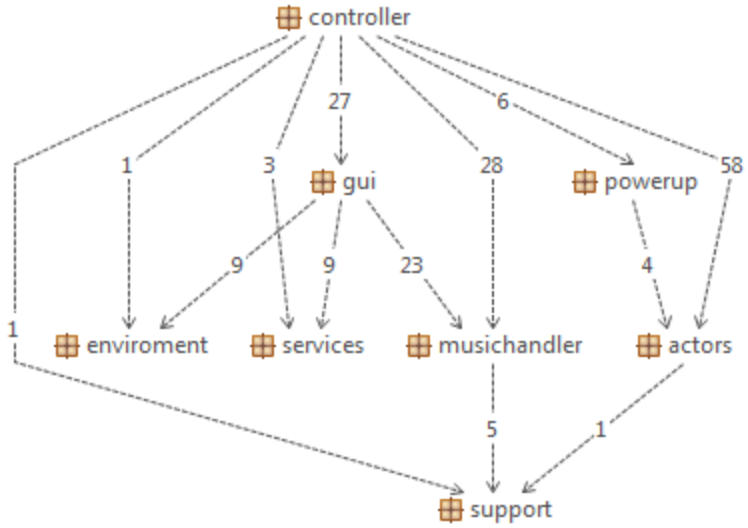

Figure.12312 High level design

2.3 Concurrency issues

N/A

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