System design document for Visualista

Group 11

Table of Contents

Version: 2.0

Date: 2014-05-23

Author: Erik Risfelt, Pierre Krafft & Markus Bergland

This version overrides all previous versions.

1. Introduction

1.1. Design Goals

The application's design must be loosely coupled, allowing for the GUI to be switched with ease, and to allow easily future modifications of subsystems. The design should be fully testable, and not be depending on any libraries other than XStream. The design should not contain any code borrowed from any graphical libraries, and the graphical backend should be easily switched.

The design should be isolatable and each partition should be testable, and function within its own borders.

1.2. Definitions, Acronyms and Abbreviations

All terms and definitions used within the Visualista application are found below. If any terms or definitions would be found on another source, do refer to those written within this document as the terms and definitions used for this project. This applies to all terms and definitions, but in particular the definition "Visual Novel" following below.

- Visual Novel - A graphical interpretations of a story defined by a user, with limited user interaction and in most cases support for changing the story clientside. To be compared with a single player game without a goal, hence not meeting the real criteria for being defined as a game.

- GUI - Graphical user interface
- Java - Platform independent programming language
- JRE - Java Runtime Environment. The additional software required to run Java applications.
- MVC - Model-View-Controller. A code technique to avoid mixing of model and view code, with a middle part referred to as the controller.

- Novel - A gathering of multiple scenes. Could be compared with a project within other applications. A novel is the highest member in the hierarchy of a Visual Novel and contains the Metadata for all other members of the hierarchy.
- Scene - A scene contain a grid, a text field as well as a background image. A novel contains at least one scene, and a scene's grid is at least 1x1 large, but is not required to have a defined image or text. When a Visual Novel is played, one and only one scene can be active at a time, and is what is shown to the user.
- Grid - A grid is a *n* x *n* graphical interpretations of tiles, and is contained by a scene. A grid is always square, and need to be at least 1x1. Without a grid, a scene cannot exist. A grid holds no data on what its tiles contains, and have no actions bound to it.
- Tile - A tile is a single square with a defined size, currently set as 32 x 32

pixels, and contains an actor. Tiles may not under any circumstances lack an actor, and the definition “empty tile” is in fact a tile holding what is defined as the Shell Actor. A tile is responsible for pass on when it is clicked by the user, and to hold the data of its actor.

- Actor - An actor is a graphical representation of an object, and carries a number of actions. An actor is defined as placed once it is represented in a tile, and is otherwise uncallable. Actors are not responsible for supplying the data to the director, but are responsible for handing the actions to a tile upon request.
- Action - An action is contained within the actor, defined by the user creating or editing and can do one of the four following things:
    - Clear a tile. This is defined as setting the current actor in that tile to the Shell Actor.
    - Set the actor of a tile at a specified location to a predefined actor. The location may only be specified as an exact value, or the value of the tile containing the actor containing this action.
    - Update the text area of a predefined scene. This may be updated to an empty string, effectively clearing it. To see the definition of the text area, see the GUI sketch.
    - Change the current scene to another scene. This dynamically saves the current state of the current scene before changing.

2. System Design

2.1. Overview

This application will use the standard MVC model, with and without event handling. The application features two different GUIs, and with that two different MVC parts. These will be referred to as the Editor respective the Player.

2.1.1. The Editor Model Functionality

The model supplied for the Editor will have its exposure handled via a direct reference to the Visualista object. The visualista object in turn supplies exposure to the model, both via interfaces and direct references. The visualista object it the top level reference, see Figure 1.

2.1.2. The Player Model Functionality

The model supplied for the Player will have its exposure handled via a direct reference to the Visualista object. The visualista object in turn supplies exposure to the model, both via interfaces and direct references. The visualista object it the top level reference, see Figure 1. This is the same as the Domain Model.

2.1.3. Event Handling

The controller supplies logic and data upon request, which is handled via events tuned for the specific task. The events carry with them data about what they are about to reach, and data what they are reaching for. This is due to the many different events which can handle during run and the fact that most of these events carry the same basic needs. This leads to few event classes which all have the ability to carry large quantities of data. While this approach has left the program with efficient coding, it does cause typecasting without checking and possibly faulty code when extensions are implemented.

2.1.4. Object Hierarchy

To keep all relevant objects in order and under surveillance at all times, each object in a Visual Novel falls under a subcategory, which is explained in the during section 1.2. Definitions, Acronyms and Abbreviations. With this structure data is easy to obtain, and exposure easily regulated with interfaces, methods and references. This also allows lossless data handling, with no risk of losing track of objects other than the object highest in the hierarchy. In this case, that object is the visualisa object, which in turn is supplied to the controller. Losing track of this object would mean losing track of the controller. This in turn means losing track of the program, and would mean a fatal bug has been found.

2.1.5. User Defined Variables

In the application there is a large amount of variables which are defined by the user mid run. This leads to a need to make sure these variables can't be set to any values which in turn would prove fatal for the program. In all methods handling user defined data, a failsafe is in progress. For example, this means that text strings will be replaced by an empty string, would the user try to input a null value, and images are handling null events by replacing them with an empty image.

2.1.6. Saving and Loading

The application supports saving and loading during running the application. This is handled by the io package, and renders depending on the Visual Novel loaded. Loading a novel resets the program in all other definitions, which means that any events currently in actions are interrupted. This should not be a problem, as there are no actions handling delayed and/or increasingly long chains of actions.

2.2 Software decomposition

2.2.1. General

The application is composed into the following modules, which are visually represented in Figure 2.

- core - Holds classes which are referenced in variables reached on a global scale.
- editorcontroller - The controller for the editor view. Controller part of the MVC for the Editor.
- io - The module responsible for reading and writing to and from files.
- model - The model objects. Model part of the MVC for both the Editor and the Player.
- playercontroller - The controller for the player view. Controller part of the MVC for the Player.
- util - Utility classes used by objects across the application
- view - The views for both the Editor and the Player. View part of the MVC for both the Editor and the Player.

2.2.2. Subsystems

The main method is run via the visualista-desktop subsystem, due to restrictions from the graphics library.

2.2.3. Layering

Indicated by Figure 3.

2.2.4. Dependency Analysis

Indicated by Figure 3. No faulty or circular dependencies.

2.3. Concurrency Issues

NA. No more than one thread are run, and in case a second or more threads would be added, events are synchronised.

2.4. Persistent Data Management

All persistent data will be saved in .xml files, and all persistent data is in the form of user defined Visual Novels. The application itself have no dependencies when it comes to persistent data.

2.5. Access Control and Security

NA

2.6. Boundary Conditions

NA

3. References

MVC - See http://en.wikipedia.org/wiki/Model-view-controller

Appendix

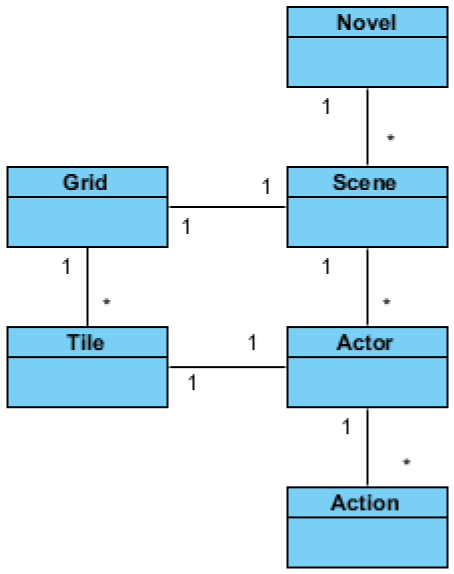

Figure 1

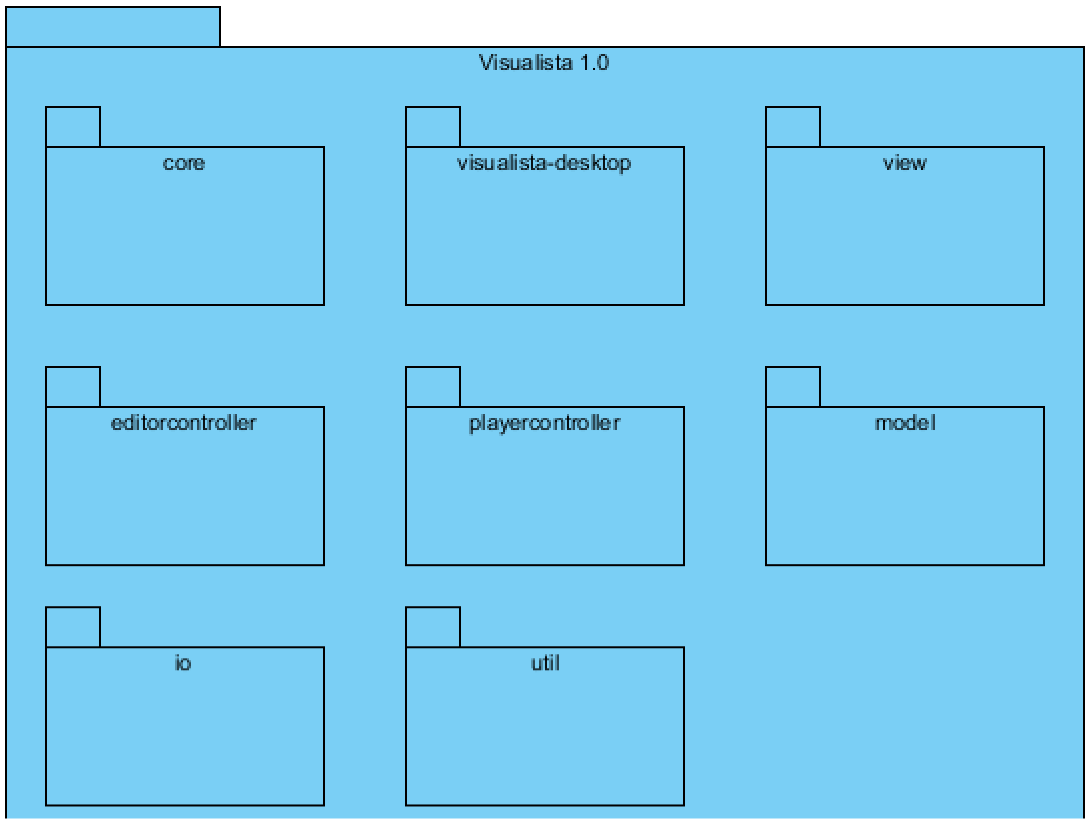

Figure 2

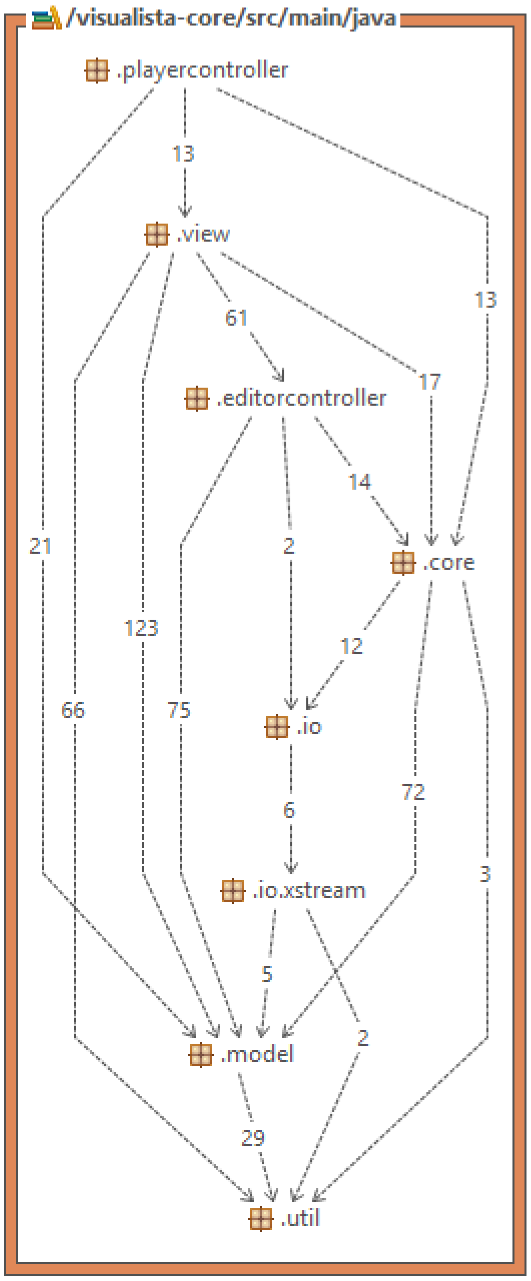
/visualista-core/src/main/java
.playercontroller
.view
.editorcontroller
.core
.io
.io.xstream
.model
.util
13
13
61
17
14
21
2
123
12
66
75
6
72
3
5
2
29

Figure 3