Nappy, the ingenious
Software Architecture Document

Revision History

Date	Version	Description	Author
12/11/15	1.0	First version	Mehmet Ali Incekara
28/11/15	1.1	Add SRS ref.	Mehmet Ali Incekara
19/12/15	1.2	Updated all Pictures and Added DB Model	Marc Mahler
19/12/15	1.3	New structure	Mehmet Ali Incekara
21/12/16	1.4	Update UCD (grammar, style, …)	Mehmet Ali Incekara
02/05/16	1.5	Updated Classdiagramm	Marc Mahler
10/05/16	1.6	Fix SAD and add pattern	Mehmet Ali Incekara
12/05/16	1.7	Add new Database Diagram	Mehmet Ali Incekara
19/05/16	1.8	Add final state pattern diagram	Mehmet Ali Incekara
03/06/16	1.9	Add deployment view	Mehmet Ali Incekara

Table of Contents

1. Introduction 4
   - 1.1 Purpose 4
   - 1.2 Definitions, Acronyms, and Abbreviations 4
   - 1.3 References 4
2. Architectural Representation 4
3. Architectural Goals and Constraints 4
4. Use-Case View 4
5. Logical View 5
   - 5.1 Overview 5
   - 5.2 Separated view 6
   - 5.3 State Pattern 7
6. Process View 8
7. Deployment View 9
8. Implementation View 9
9. Data View 9
10. Size and Performance 9
11. Quality 9

Software Architecture Document

1. Introduction

1.1 Purpose

This document provides an architectural overview of Nappy, the ingenious in aspects of different architectural views.

1.2 Definitions, Acronyms, and Abbreviations

(n/a)

1.3 References

SRS	https://github.com/nappydevelopment/docs/blob/master/pdfs/Software%20Requirements%20Specification.pdf
Test Plan	https://github.com/nappydevelopment/docs/blob/master/pdfs/Test%20Plan.pdf
SonarQube	http://193.196.7.25/overview?id=5235
Coveralls	https://coveralls.io/github/nappydevelopment/Nappy-the-ingenious?branch=master
Codacy	https://www.codacy.com/app/NappyDevelopment/Nappy-the-ingenious/dashboard

2. Architectural Representation

The project Nappy, the ingenious will use the MVC-principles. We don’t use a framework because we are using JavaFX. We implemented our own MVC. Every GUI has an own Controller and Resource.

3. Architectural Goals and Constraints

The main goal of this architecture is to separate the view from the logic. The view is “stupid” and knows nothing.
We will use the Framework from JavaFX for our project.

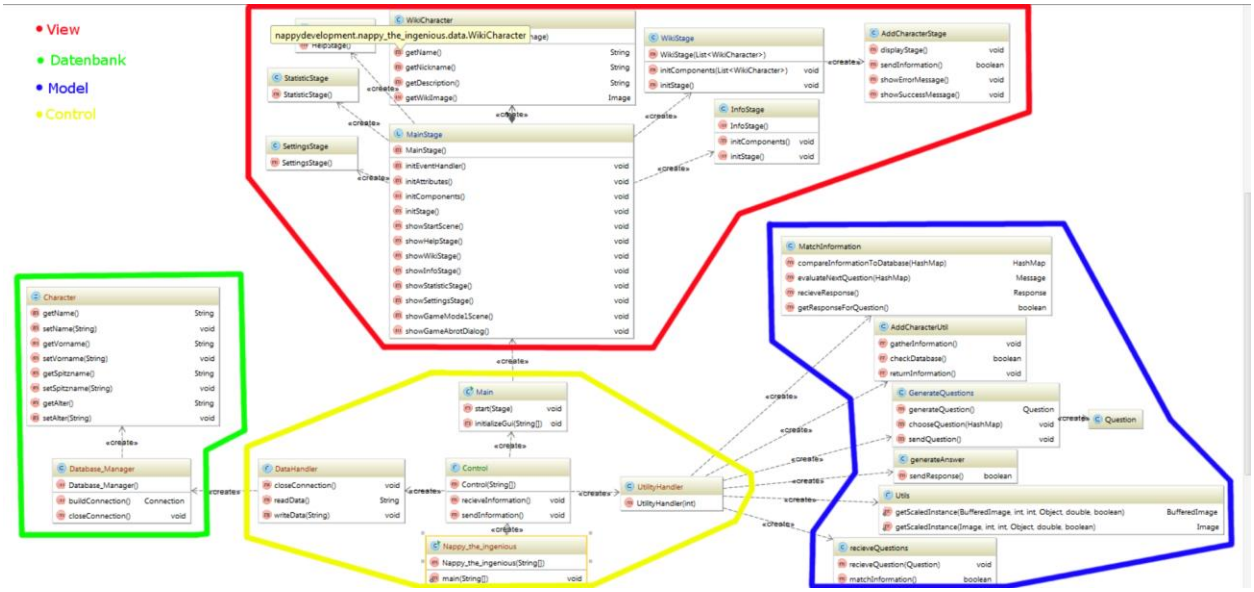
4. Use-Case View

Overall-Use-Case-Diagram:
https://github.com/nappydevelopment/docs/blob/master/pdfs/Overall%20Use%20Case%20Diagram.pdf

5. Logical View

5.1 Overview

This Class diagram represents how we did plan to design our application.



https://github.com/nappydevelopment/docs/blob/master/pdfs/Class_Diagramm.pdf
After we updated our project, there was a new class diagram produced. In the new class diagram we completed our work on the MCV principle and used the JavaFX Framework to create it.

The new Class Diagram is pretty huge, so you won't be able to read what is in there in this document. You can follow the provided link to see the full Class Diagram as an SVG and zoom and scroll in it.

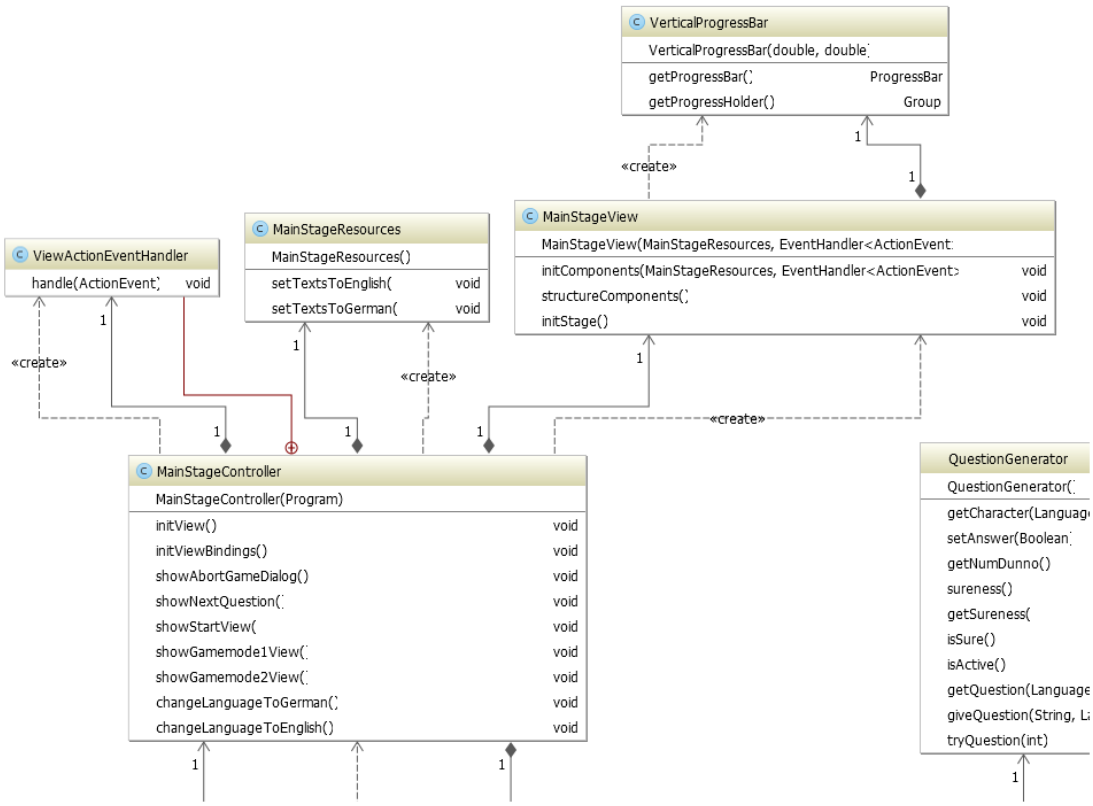
https://github.com/nappydevelopment/docs/blob/master/svg/New%20Classdiagram.svg

Our project got updated again, so we have a new Version of our class diagram. It is even bigger than the last one. Because of this, there will again only be a link to Github in this document, so you can follow the link to get the class diagram as an SVG and be able to scroll around in it. Our MVC concept did not change, so everything below can be viewed as it is.

https://github.com/nappydevelopment/docs/blob/master/class diagram/Classdiagramm_02_05_16.svg

Our MVC concept consists in many "stages" as they are called in JavaFX. A stage represents one window or one label inside a window. Every Stage has their own MVC concept, which makes it pretty hard to show a detailed view about Model, View, Control and Database. That's why we decided, to add one single stage right here and mention the important parts in the chapters "Separated View" below.

5.2 Separated view

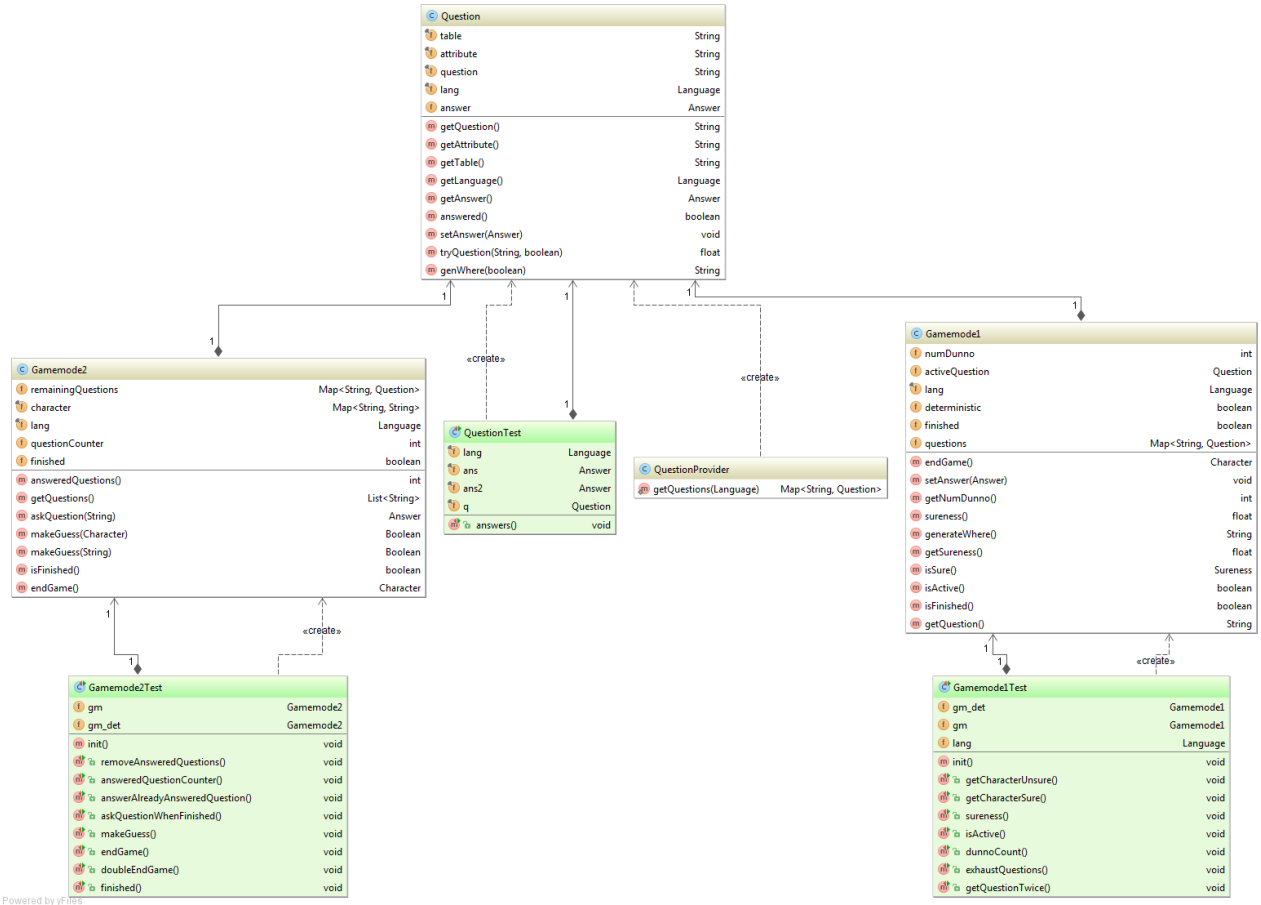


The main stage has his own controller, model and view so every stage is independent. We can easily switch every stage or add new stages.

5.3 State Pattern

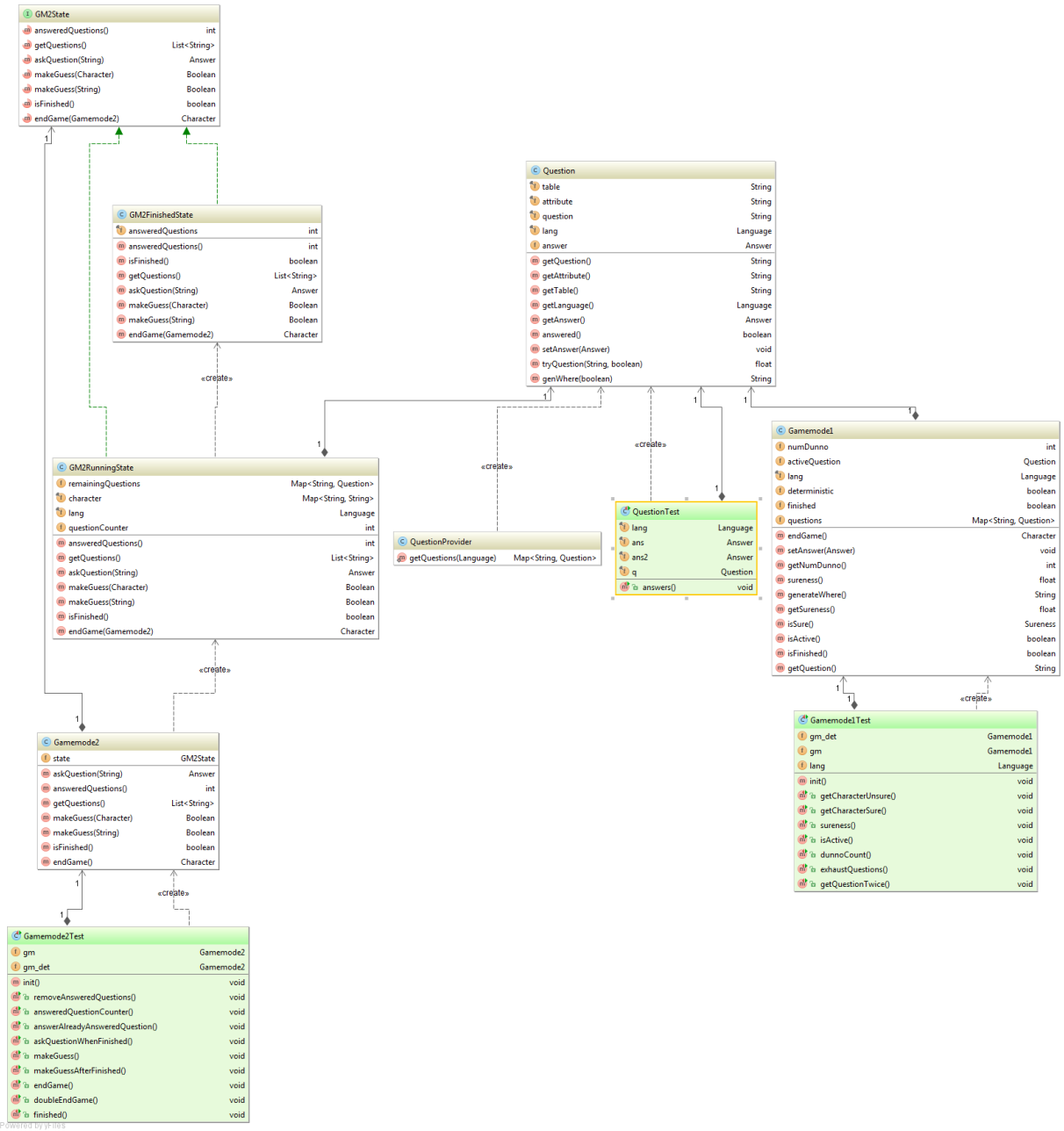
We are using the state pattern in our two game modes to improve our architecture.

Before we used the pattern the UML looks like this:



The state pattern helped us eliminate one instance variable which represented the state.

After implementing the pattern the UML looks like this:

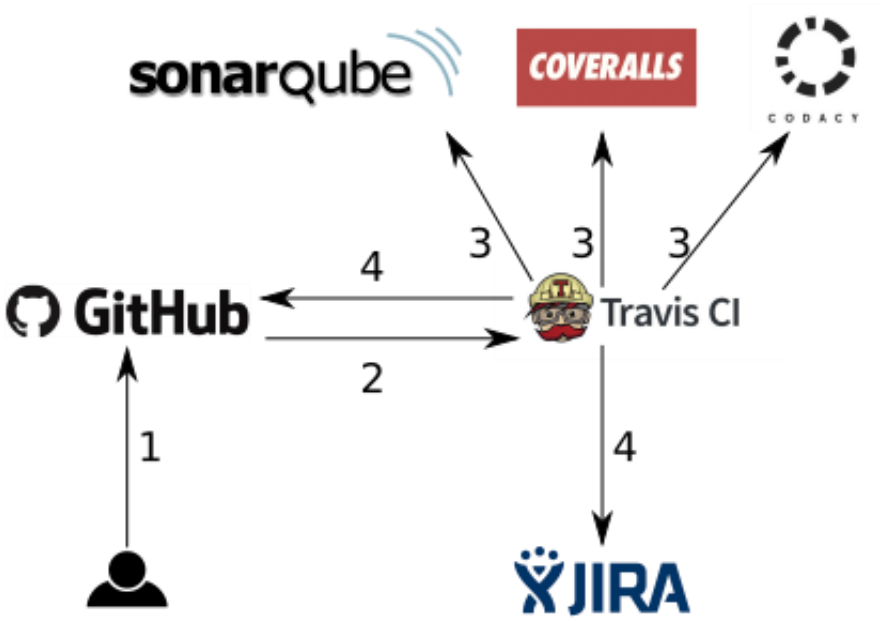


We did add the state pattern now to game mode 1 and 2:
https://raw.githubusercontent.com/nappydevelopment/docs/master/state-pattern/gm1u2.png

6. Process View

(n/a)

7. Deployment View



1. The User pushes his changes to master on GitHub.
2. Travis-Ci will start the deployment. Here you can see our travis.yml.
3. Travis-Ci will update our coverage and metric tools Coveralls, SonarQube and Codacy.
4. After success Travis-Ci will push a new JAR to GitHub where everyone can download the game. If the deployment fails, Travis-Ci will create a Jira Issue.

8. Implementation View

(n/a)

9. Data View

For our Database, we also have a model that might give you a short look at our Database-structure:

https://github.com/nappydevelopment/docs/blob/master/pdfs/Database%20Diagram.pdf

The database structure is too big to add it to this document.

10. Size and Performance

The size and the performance are really important for our project. We are trying to keep the size of the project as small as possible. Also we will load our database into the memory to increase the performance. So we will need a few seconds until the program starts to load all pictures and the database.

11. Quality

The quality of our project Nappy, the ingenious is also important. We used a lot of sources to get all information about the characters and so our database has a stable foundation.