

Grey Matter

System Design Document

and I OOPP:
ALINE EIKELAND
HUGO GANELIUS
VIKTOR JOHANNESSON
EMIL LUNDQVIST
FELIX OLIV

1 Introduction

This system design document intends to introduce the reader to the brain test mini game android application that is Grey Matter. The document will develop on design choices and process, system structure and qualities as well as the finished product.

1.1 Definitions, acronyms, and abbreviations

Gamification - applying gameplay principles on non-gaming activity to increase attractiveness

JSON - JavaScript Object Notification, easily readable file format for data storage

JSON-server - JSON module to easily mockup servers without having to program server part.

GSON - Google's JSON server service.

MVC - Model View Controller pattern.

MVVM - Model View ViewModel pattern.

Normal distribution - Natural probability calculation, looks like a bell curve.

Standalone application - A application that runs locally on the device and does not require anything else to be functional.

2 System architecture

GrayMatter is a mostly standalone application, and all the logic is therefore built into the app. However to be able to compare your score with other players, the app needs a internet connection to download other players data which is located on a server.

Grey Matter application contains all necessary logic and GUI. The social interaction and leaderboard features however requires the need of a database and server. If the player experiences infrequent internet connection, the offline features can still be used although the results will not be updated until the player retains internet access.

2.1 Flow of the application

When the application is started the method on-Create in the Main-activity class is called. This method will instantiate the main-page XML file which displays the main-page to the,

3 System design

3.1 System packages

3.1.1 Social

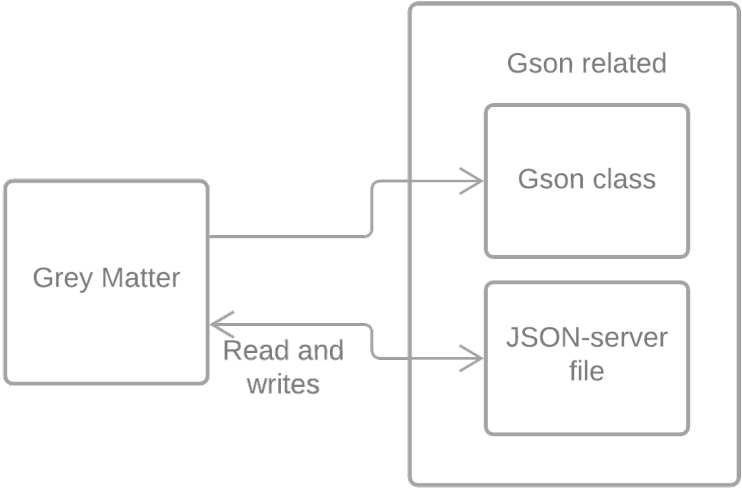


Figure 1: External use of Gson in social package

The social

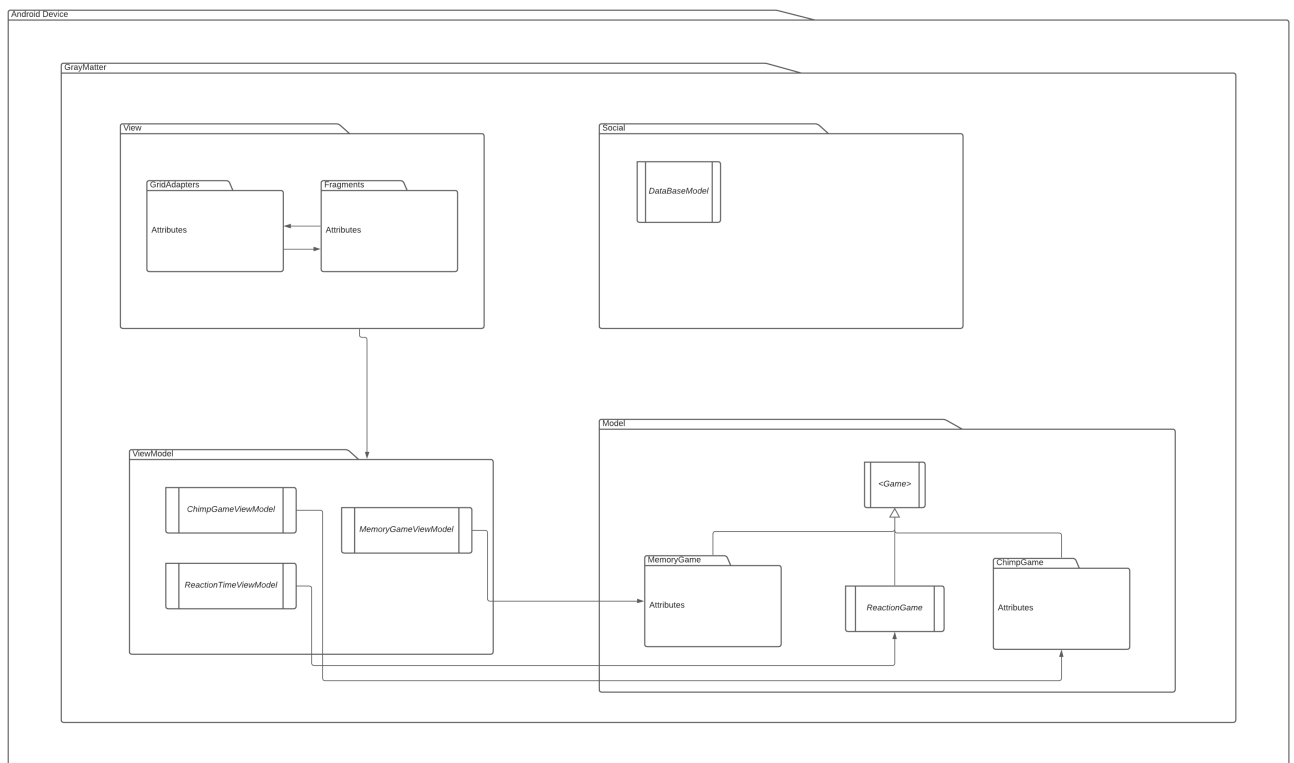


Figure 2: The package model of the application

3.2 Modular

3.2.1 MVC-pattern

3.2.2 MVVM-pattern

When the View is created, onClick events are also created, to get a response from the user. If something is pressed the View signals to the ViewModel that something has been clicked, which signals to the model that some values have changed. Then the model tells its observers to update with the new directives, and then the same cycle repeats. This is to not make the model communicate with other parts of the program except itself.

Viktors tankar: Vi använder oss inte av singleton för att inte begränsa oss något (?)

Vi använder inte trådar för det var ett externt bibliotek (?)

I möjligaste mån försöker vi att använda oss av kopior eller andra metoder som private eller final för att minska muterbarhet

Adapter patten? (göra annars inkompatibla komponenter kompatibla) för att kunna ha ett gridview i ett fragment

Facade Pattern (öka abstraktionen genom att gömma intern komplexitet): Detta gör vi med game klassen för att minska beroenden mellan VM och modellen

Composite pattern (använda en grupp av objekt på samma sätt som ett objekt)

Iterator patter (vet knapopt vad det innebär)

State pattern: vi använder oss av states i form av starta och stoppa spel

Template method pattern (när en del av kod är gemensam, bryt ut den del som är gemensam i en abstrakt klass): i game som är en abstrakt klass har vi brutit ut

observers som alla spel ska använda sig av

Slut på Viktors tankar

3.3 One purpose

Assigning a single responsibility to a package or class goes hand in hand with the last subsection on modularity.

3.3.1 Data Mapper

Data Mapper is not a traditional Object oriented design pattern. Although it is based around the principle

3.4 Foolproof

By giving copies of instances and variables to other classes we can assure that mutation does not happen in places we do not want it to happen, for the same reason attributes and variables get assigned private and final.

3.5 Every edge case noticed

3.6 High abstraction

4 Persistent data management

Grey Matter uses a JSON server for storing user data and game records.

5 Quality

5.1 Testing

The application is tested with jUnit tests and Travis automatic build testing. The project can be viewed continuously at [this github repo](#).

Following custom testing has been written:

- Tests on user profile and security
- Tests on social interaction such as adding and removing friends
- Tests on minigames

- Building the game
 - Playing successfully
 - Playing and failing or ultimately losing
 - Breaking the game through unusual play patterns

5.2 Issues

The code has mostly healthy dependencies. The social package has a file, PlayerMapper, which needs to be separated into smaller areas of responsibilities. It is too long and it not following the DataMapper pattern as intended. When losing in memory game one does not...

5.2.1 STAN results



Figure 3: The dependencies in GrayMatter from the application STAN

Diagrams

The STAN diagram shows the dependencies between the Model, View, and ViewModel.

There are no circular dependencies except for the one between adapters and the ViewFragments. These dependencies is necessary because the adapter needs to know what to print on the screen when the grid is initiated. The View is the object

being created first, and then needs a grid (varför gridAdapter inte finns i fragment vet jag inte (Felix)) that needs to communicate with the Fragment when a grid is pressed.

The ViewModels takes care of the communication to the model, and uses the functionality available. It then stores the values and directives for the Fragment to use. The dependencies between Fragments and the ViewModel is also one-way, the Fragments can use the ViewModel without the need to be seen. This makes the coupling low and enables extension with ease.

The games all have dependencies to the model, to use the functionality. These dependencies are necessary to use the functionality, and the important part is that the model does not talk to any other parts of the program, which it does not.

EventBus (ingen aning hur det funkar)

5.2.2 PMD results

PMD test found problems mostly related to small fixes as failure to remove unused code and imports, extra brackets of different kinds and misadaptation to Java conventions, such as capital letters in the beginning of a package name.

It did however also highlight following mistakes in the code:

- Typing to implementation class instead of interface - this limits flexibility.
- Using unnecessary local variables where one could immediately return result - this clutters the code and uses additional storage.

For full PMD analysis see appendix.

5.3 Access control and security

Grey Matter uses personal accounts for storing results and social interaction. The system is developed for this application and the user

6 Appendix

7 References

[Gradle documentation](#)

[Gson documentation](#)