Build 1 Risk Game Document

Team 14

Group members:

Yann Kerichard 40059813

Yueshuai Jiang 40046639

Che-Shao Chen 40055392

Contents

[1. Structural design 4](#_Toc529561148)

[1.1 Controller 4](#_Toc529561149)

[1.1.1 MainController 4](#_Toc529561150)

[1.1.2 GameController 4](#_Toc529561151)

[1.1.3 MapEditorController 4](#_Toc529561152)

[1.2 Model 4](#_Toc529561153)

[1.2.1 GamePlay 4](#_Toc529561154)

[1.2.2 Map 4](#_Toc529561155)

[1.2.3 Utilities 5](#_Toc529561156)

[1.3 View 5](#_Toc529561157)

[View is the component that interact directly with players and model, acting as an interface to show players the information and possible option they can choose. 5](#_Toc529561158)

[1.3.1 Common 5](#_Toc529561159)

[1.3.2 GamePlay 5](#_Toc529561160)

[1.3.3 MapEditor 5](#_Toc529561161)

[2.diagram of the architecture design 6](#_Toc529561162)

[3.Code Standards 7](#_Toc529561163)

[3.1 Naming should reveal the schematic 7](#_Toc529561164)

[3.2 Name should in right length 7](#_Toc529561165)

[3.3 Same concept should be named in same word 7](#_Toc529561166)

[3.2 Naming by using words related to the domain when naming 8](#_Toc529561167)

[3.3 Method and Attribute names start with a lowercase letter and use uppercase letters to separate words 8](#_Toc529561168)

[3.4 Constants use upper case letters with underscores between words 9](#_Toc529561169)

[4. Code Layout 9](#_Toc529561170)

[4.1 Use the line break wisely 9](#_Toc529561171)

[4.2 Indent style 9](#_Toc529561172)

[4.3 Blank lines 10](#_Toc529561173)

[4.4 Use of space 10](#_Toc529561174)

[5. Commenting 11](#_Toc529561175)

[5.1 Block Comments 11](#_Toc529561176)

[5.2 Single-Line Comments 11](#_Toc529561177)

[5.3 Trailing Comments 11](#_Toc529561178)

[5.4 Documentation Comments 11](#_Toc529561179)

[6. How you came up with a list of potential refactoring targets? 12](#_Toc529561180)

[7. References 13](#_Toc529561181)

The Design of The Architecture of The Risk Game

# Structural design

In our project, we are using the MVC architecture design. The structure separates into 3 main parts for the system, called Controller, Model, and View.

## 1.1 Controller

Controller is the for interface between view and model, receiving input from user and initiate response by calling corresponding objects. After receiving the inputs, the controller will call other models to complete the process according to the input. In the risk game, we have made 3 controllers, they are GameController, MainController, and MapEditorController.

### MainController

In the MainController, it is the menu model of the system when first launched, it gives players 2 options at the beginning, either play game or edit map.

### GameController

In the GameController, there is one method , excute , to perform the whole phase of the gameplay, including startup phase, reinforcement phase, attack phase and fortification phase.

### MapEditorController

This controller will be called when the player choose to edit map. Any operation related to map editing, will be implemented in this controller.

## Model

Model is the component where all the data related logic respond to, the model will manage all behavior and data during the whole process of the application, including responding request to access the state of data from view, and change state request from controller.

* + 1. GamePlay

In the GamePlay package, we created classes and their related attributes that will be needed during the game , which includes card, dices, phase and player.

### Map

The Map package contains classes like Continent.class, Country.class, Map.class, and MapEditor.class. The first two classes deal with the basic data and logic for two main map properties in the game. The Map class is one of the most important class in the project, since no operation can run without map, it carries the most functions among other classes, which includes functions for checking validation of maps, data change, saving, and loading maps, set country armies and so on. The functions in MapEditor class can let users create, editing, and save map files.

### Utilities

Utilities package includes useful classes that can be widely reused by other classes. Such as FileHandler class for reading and writing files. Message class for showing messages to the user. Random class dealing with random index values. The StringAnalyzer class focus on the string function logic.

## View

## View is the component that interact directly with players and model, acting as an interface to show players the information and possible option they can choose.

### Common

The Common package contains the most generalized views in the game, which includes View, MainMenuView, MapSelectionView.

View is an abstract class which contains the input validation.

MainMenuView is the main menu interface

MapSelectionView is the interface when player is selecting maps.

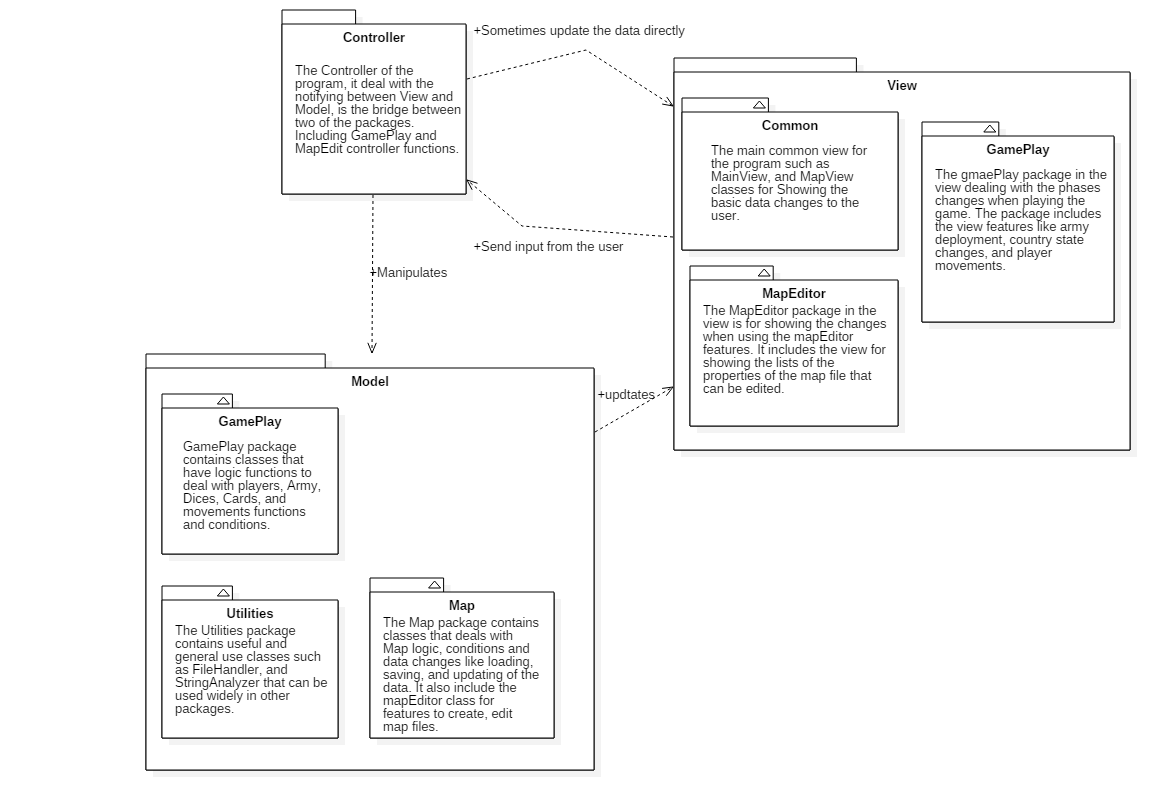
### GamePlay

The gamePlay package contains all interfaces during the game play, which includes startup, reinforcement, attack, fortification, phase change, card exchange, map viewing, world domination view, winner view.

### MapEditor

The MapEditor class fulfill functions for showing the map properties to help users create and editing maps.

# 2.diagram of the architecture design



# 3.Code Standards

Normative naming can make the code more readable.

## 3.1 Naming should reveal the schematic

bad:



good:



## 3.2 Name should in right length

bad:

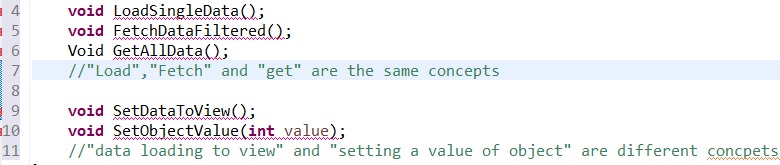


good:

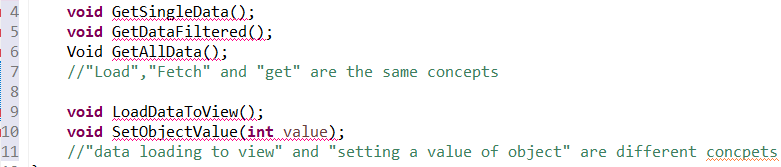


## 3.3 Same concept should be named in same word

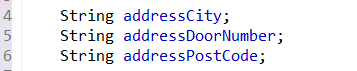
bad:



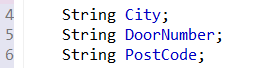
good:



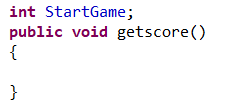
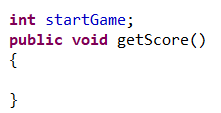
## 3.2 Naming by using words related to the domain when naming

bad:  


good:



* 1. Method and Attribute names start with a lowercase letter and use uppercase letters to separate words  
     bad:

  
good:  


## 3.4 Constants use upper case letters with underscores between words

bad:



good:  


# 

# 

# 4. Code Layout

## 4.1 Use the line break wisely

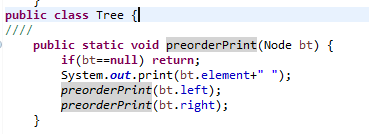
Breaking the code lines only when: The statements exceed the column limit, or,

One statement is over.

## 4.2 Indent style

Indent size is 4 columns, nested continuation indent may add 4 columns at each level

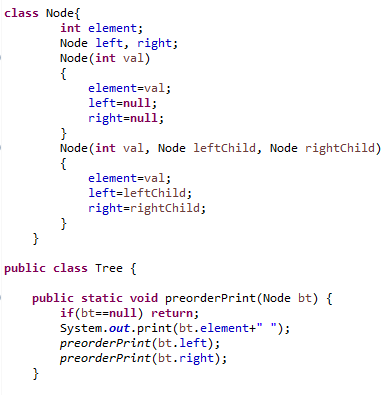
and always use tab instead of space



## 4.3 Blank lines

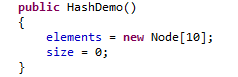
Blank lines can be added between major sections of a long and complicated function,

between public, protected, and private sections of a class declaration, between class declarations in a file , and, between function and method definitions.



## 4.4 Use of space

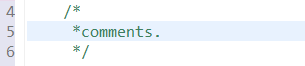
use space between operators and equals.



# 5. Commenting

## 5.1 Block Comments

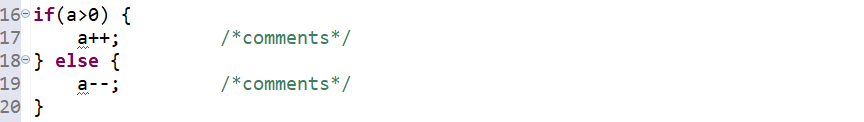
The block comment should be preceded by a blank line and have an asterisk “\*” at the beginning of each line. The block comments usually appear at the beginning of each file and before each method. If the block comments inside a function or method then it should be indented to the same level as the code they describe.



5.2 Single-Line Comments  
Short comments can appear on a single line indented to the level of the code that follows. If a comment can’t be written in a single line, it should follow the block comment format.

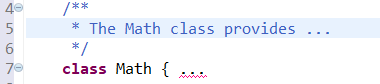


5.3 Trailing Comments  
The comments can appear on the same line as the code they describe if they are very short, but they should be set apart from the code. If more than one short comment appears in a code block, they should all be indented to the same tab setting.



## 5.4 Documentation Comments

Documentation comments should be set inside the comment delimiters /\*\*...\*/ and appear before the declaration.

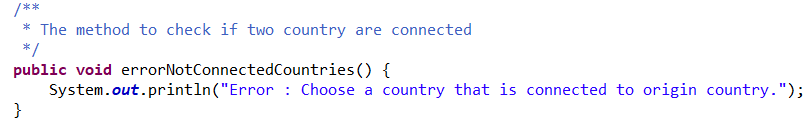
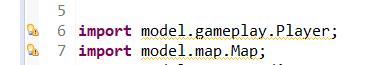


# 6. How you came up with a list of potential refactoring targets?

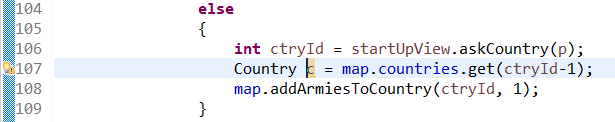
We executed the following process just after the release of build 2, and before beginning to implement the new requirements:

* Inspecting most **important logical units**
* Inspecting **requirements asked**
* Analyzing and inspecting classes having **relationships** with other
* **Defining responsibilities** and discussing about what each class should be able to do or not to do. Then, we compared with the actual responsibilities of the classes in our project
* Looking inside each class and to ensure the **methods names** quickly give an idea of what is going on
* Finally, by inspecting each file in each package and checking if **condition statements** could be refactored. Also checking warnings in Eclipse allowed us to remove/change unused lines of code. All refactoring operations were hitting parts of the system for which tests were designed.
* After refactoring, we launched test suites.

List of 3 refactoring operations :

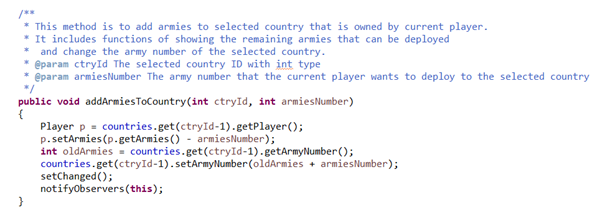
* fortificationView.errorNotConnected() becomes fortificationView.errorNotConnectedCountries();  
   
* Removing unused imports everywhere  
    
  

Unused imports removed in MapEditorController

* Removing unneeded computations  
    
  

Unneeded variable and computation in GameController class

Moving method to other location



Moving addArmiesToCountry method from map to country class

# 7. References

<http://google.github.io/styleguide/javaguide.html>

<https://www.oracle.com/technetwork/java/javase/documentation/codeconventions-141999.html#216>