Build 1 Risk Game Document

Team 14

Group members:
Yann Kerichard 40059813
Yueshuai Jiang 40046639
Che-Shao Chen 40055392

Table of Content

The Design of The Architecture of The Risk Game

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.

Controller:

For the design of the Controller, we create 3 different classes inside the package called MainController.class, GameController.class, MapEditorController.class to deal with the control for the main menu, gameplay, and the map editor. Each of the class focusses on the specific area of the features.

Model:

Packages: GamePlay, Map, Utilities

The model package mainly focuses on the implementation of the logic functions of the gaming and editing part.

In the GamePlay package, we implement Army.class for the army battles. Card.class for getting and trading cards. The Player.class for data changes of each player.

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 includes functions for the data change, saving, and loading on the map. The functions in MapEditor class can let users create, editing, and save map files.

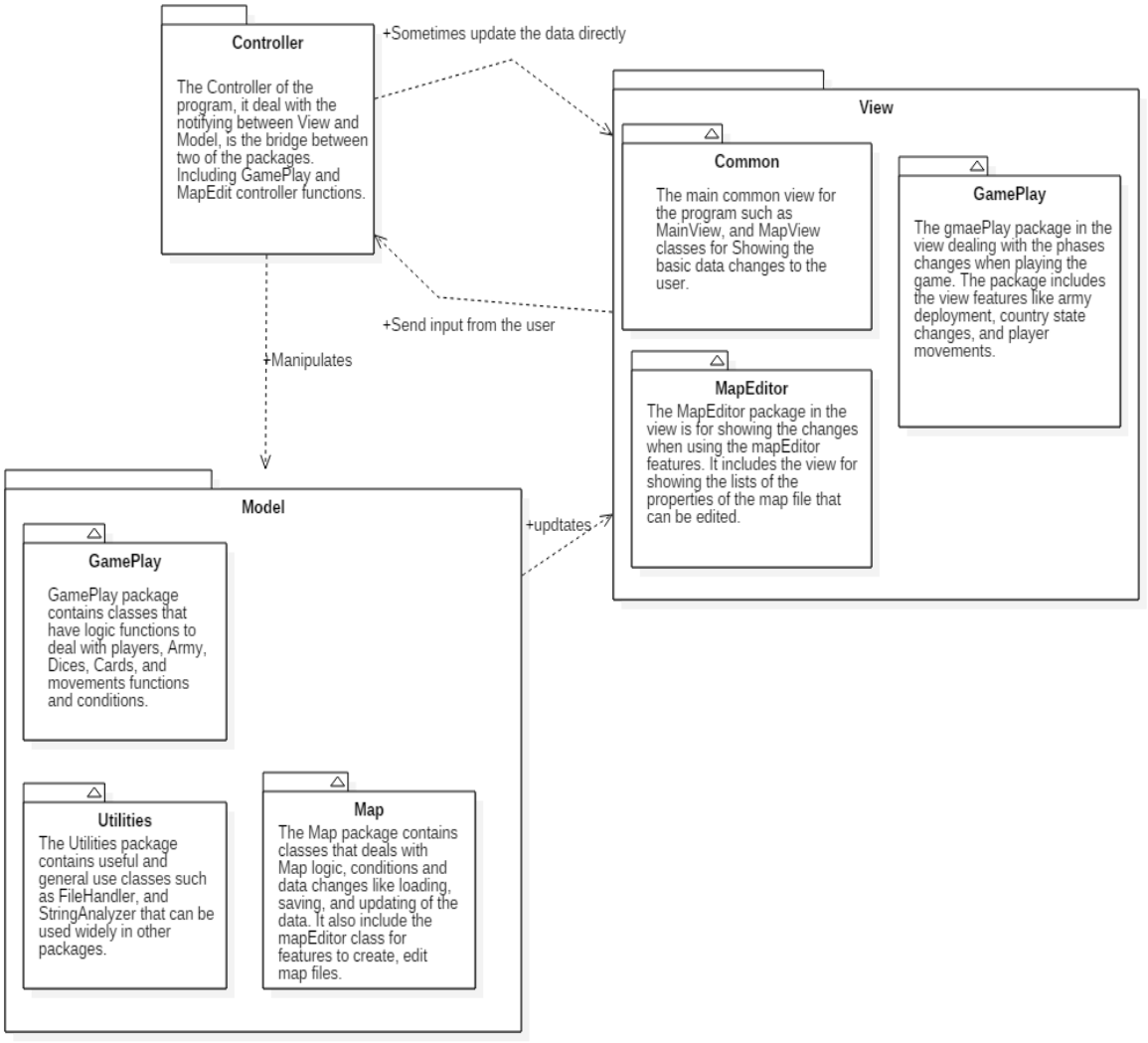
Utilities package includes useful classes that can be widely reused by other classes. Such as FileHandler class for reading and writing files. Message class for showing messages to the user. Random class dealing with random index values. The StringAnalyzer class focus on the string function logic.

View:

Packages: Common, GamePlay, MapEditor

View mainly deals with functions that making the functions in other parts can be easily viewed by users. The Common package helps to show the main menu to help users to choose whether they want to edit the map or play the game. The gamePlay package helps users to choose map files and implementing the game state phases for users to know what is happening when playing the game. The MapEditor class fulfill functions for showing the map properties to help users create and editing maps.

diagram of the architecture design



Code Standards

Normative naming can make the code more readable.

1. Naming should reveal the schematic.
   bad:

```
4      int d;
5      //d means the number of days
6
```

   good:

```
4      int daysSinceCreation;
5      int daysSinceModify;
6
```

2. Avoiding misleading information when naming.
   bad:

```
4      Teacher[] teacherList;
5      //teatcher list is an array not a list
6      Table theTable;
7      //"the" is a misleading word
```

   “theTable” is a table of teacher.
   good:

```
4      Teacher[] teachers;
5      Table teachers;
```

3. Name should in right length.
   bad:

```
4      var theTeachersListWithAllTeachersInclude;
5      var list;
6
```

   good:

```
4      var AllTeachers;
5      var TeachersInOrder;
```

4. Same concept should be named in same word.
   bad:

```
4     void LoadSingleData();
5     void FetchDataFiltered();
6     Void GetAllData();
7     //"Load","Fetch" and "get" are the same concepts
8
9     void SetDataToView();
10    void SetObjectValue(int value);
11    //"data loading to view" and "setting a value of object" are different concpets
```

good:

```
4     void GetSingleData();
5     void GetDataFiltered();
6     Void GetAllData();
7     //"Load","Fetch" and "get" are the same concepts
8
9     void LoadDataToView();
10    void SetObjectValue(int value);
11    //"data loading to view" and "setting a value of object" are different concpets
```

5. Naming by using words related to the domain when naming.
   bad:

```
4     String addressCity;
5     String addressDoorNumber;
6     String addressPostCode;
```

   good:

```
4     String City;
5     String DoorNumber;
6     String PostCode;
```

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

```
int StartGame;
public void getscore()
{

}
```

   good:

```
int startGame;
public void getScore()
{

}
```

7. Constants use upper case letters with underscores between words.
   bad:

```
public static final MaximumTemperature, main_WINDOWS;
```

   good:

```
public static final MAXIMUM_TEMPERATURE, MAIN_WINDOW_WIDTH;
```

Code Layout

Use the line break wisely

Breaking the code lines only when: The statements exceed the column limit, or, One statement is over.

Indent style

Indent size is 4 columns, nested continuation indent may add 4 columns at each level and always use tab instead of space

```
public class Tree {
////
    public static void preorderPrint(Node bt) {
        if(bt==null) return;
        System.out.print(bt.element+" ");
        preorderPrint(bt.left);
        preorderPrint(bt.right);
    }
```

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.

```
class Node{
        int element;
        Node left, right;
        Node(int val)
        {
            element=val;
            left=null;
            right=null;
        }
        Node(int val, Node leftChild, Node rightChild)
        {
            element=val;
            left=leftChild;
            right=rightChild;
        }
    }

public class Tree {

    public static void preorderPrint(Node bt) {
        if(bt==null) return;
        System.out.print(bt.element+" ");
        preorderPrint(bt.left);
        preorderPrint(bt.right);
    }
```

Use of space

use space between operators and equals.

```
  public HashDemo()
  {
      elements = new Node[10];
      size = 0;
  }
```

Commenting

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.

```
4       /*
5        *comments.
6        */
```

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.

```
10 if(a>0){
11
12      /*comments*/
13      . . .
14 }
```

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.

```
16 if(a>0) {
17      a++;          /*comments*/
18 } else {
19      a--;          /*comments*/
20 }
```

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

```
4      /**
5       * The Math class provides ...
6       */
7      class Math { ...
```

References

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

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