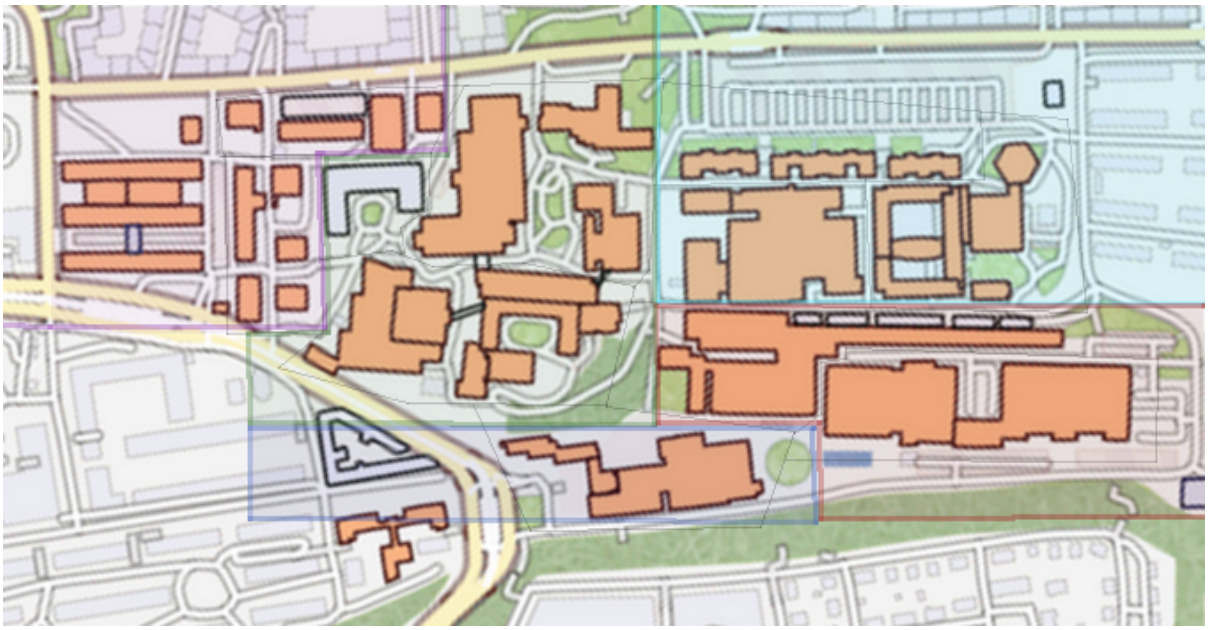
System design document for ”Chalmers Risk”



Version: 4.00

Date: 30/5 - 2015

Authors: Malin Thelin, Oskar Rutqvist, Björn Bergqvist, Robin Jansson

This version overrides all previous versions.

Table of contents:

1 Introduction

1.1 Design goals

To support testing the software must support the standards of object oriented programming, it must be possible to test an isolated part of the program.

1.2 Definitions, acronyms and abbreviations

- GUI, Graphical User Interface.
- Java, the programming language used to produce this application.
- JRE, Java Runtime Environment. This is necessary to be able to run the application. Java applications are run via this platform.
- MVC, Model - View - Controller. Design pattern to be used to avoid mixing up model and controller code together as well as keeping the interface apart from these parts.
- Turn, The application is a turn based game with two human players and thus each player will take turns performing his/her actions.
- Card, event cards and will be awarded to players who conquered a territory on their previous turn. Event cards will impact the game in various ways, granting troops to the player or ending the game prematurely.
- Phase, the game consists of three different phases, each having their own implementation and rules.

2 System design

2.1.1 Overview

The application will follow the MVC design pattern. Our implementation of MVC might be a little different than the ordinary MVC model. Our goal is to make the design as simple as possible. We want the interface to solemnly show the functions that is available for the player at the current point of time.

2.1.2 Event Cards

During the development of our event cards we choose to use an interface rather than an abstract class since we didn't have any functionality we wanted inherited and due to the restrictions against multiple inheritance in java we wanted to leave the possible option for implementation inheritance for future development.

2.1.3 Algorithms

The application uses the a path exists algorithm based upon the depth first search algorithm to find out if there exists a path of territories that are owned by the same player between two territories. The algorithm is used to check if troop movement is possible.

In order to simulate a deck of cards the application implements the Fisher-Yates shuffling algorithm. The algorithm loops through a list and moves the object to a random position out of the current position and all unvisited positions. Because of the randomization in the algorithm there is a risk that testing of it is going to yield false negatives.

When the application should display the dice during the 'Attack Phase', it was found necessary to show the dice in descending order. This is because the highest valued dice thrown by one player is compared to the highest valued dice thrown by the other. Therefore a simple bubblesort algorithm is used to sort the diceroll arrays. Since these arrays do not get larger than having at the most three different integers in them, a bubblesort works very well.

2.1.4 Phases

The application has three different phases. Each phase lets the current player do different things. To implement this each phase is represented by a different controller.

The first phase for example is the "place troop" phase. At the beginning of the phase the player recieves a number of troops depending on several factors. The player then has to place these troops on the different territories, and the player can't go to the next phase until they have placed all of the recieved troops.

For this phase we therefore have a PlaceTroopController that implements this functionality on the entities found on the GameBoard. As seen in Figure 1. Pressing a territory will place a troop there and not attack it

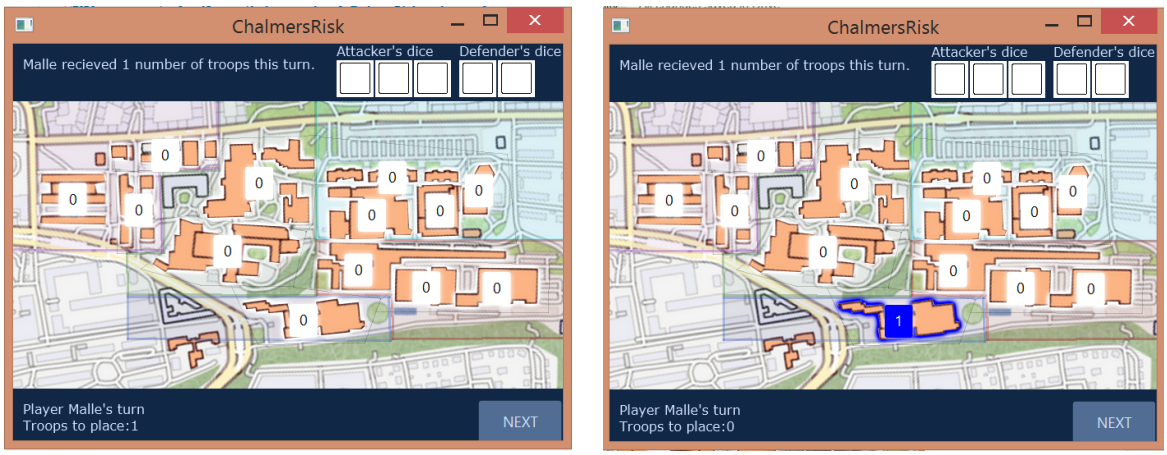


Figure 1. Entities found on the GameBoard and how they work during phase 1 - Place Troops

Furthermore, all the other phases work similarly. When they get activated a new controller is set to implement the functionality on GameBoard and so the same button will do different things depending on which phase we are in.

2.2 Software decomposition

2.2.1 General

This application is decomposed into following modules, see Figure 1.

- view, the views of the application, the main gui components, the view in our MVC model.
- gui, the smaller graphical objects are located here.
- controller, the controllers of the game are located in the package. The controllers of the MVC model of the game.
- model, this package contains all the models in our main-game. The Model in MVC.
- cardModels, the models for the different cards of our game.
- utilities, constants of the game and a sorting method.
- CRApplication, starts the game, launches the application.
- Main, calls for launch of the CRApplication.

Package diagram. For each package an UML class diagram in appendix.

2.2.2 Decomposition into subsystems

2.2.3 Layering

The layering as indicated in the figure below. Higher layers are ordered in rows from left to right with the highest in the top left corner.

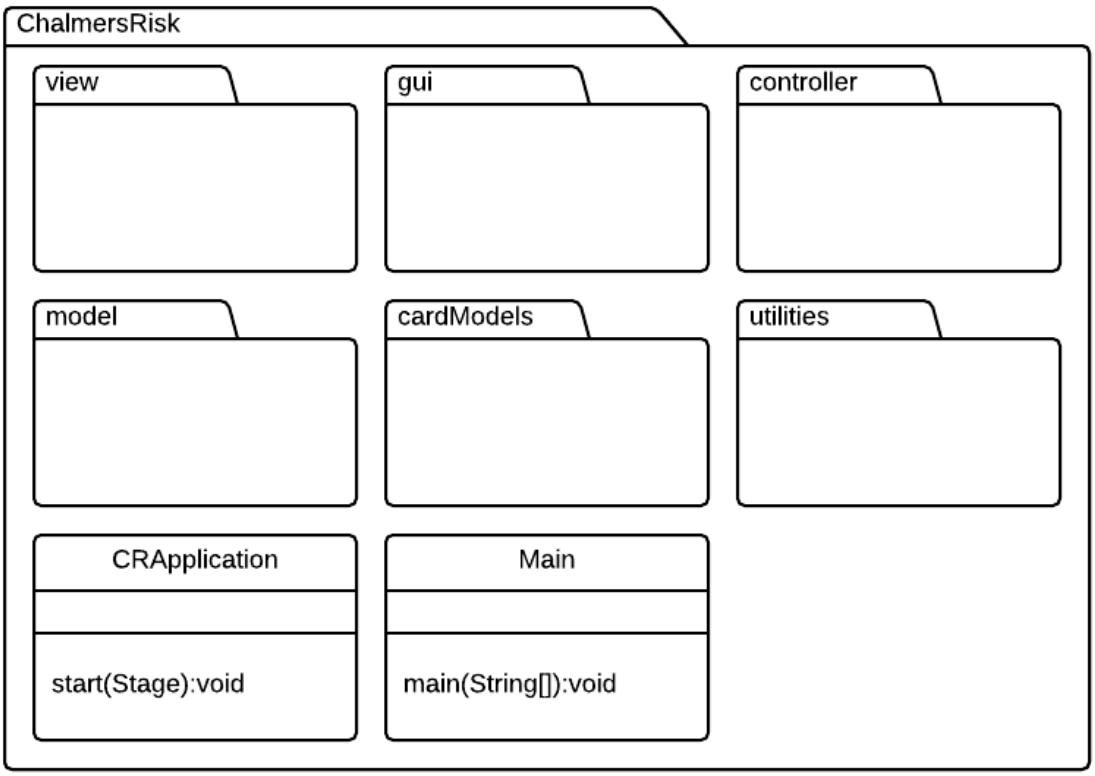


Figure 2: High level design

2.2.4 Dependency analysis

The dependency analysis is shown in Figure 3.

There are circular dependencies occuring arround utilites since for example the Territory class uses some constants like EMPTY_PLAYER in its methods.

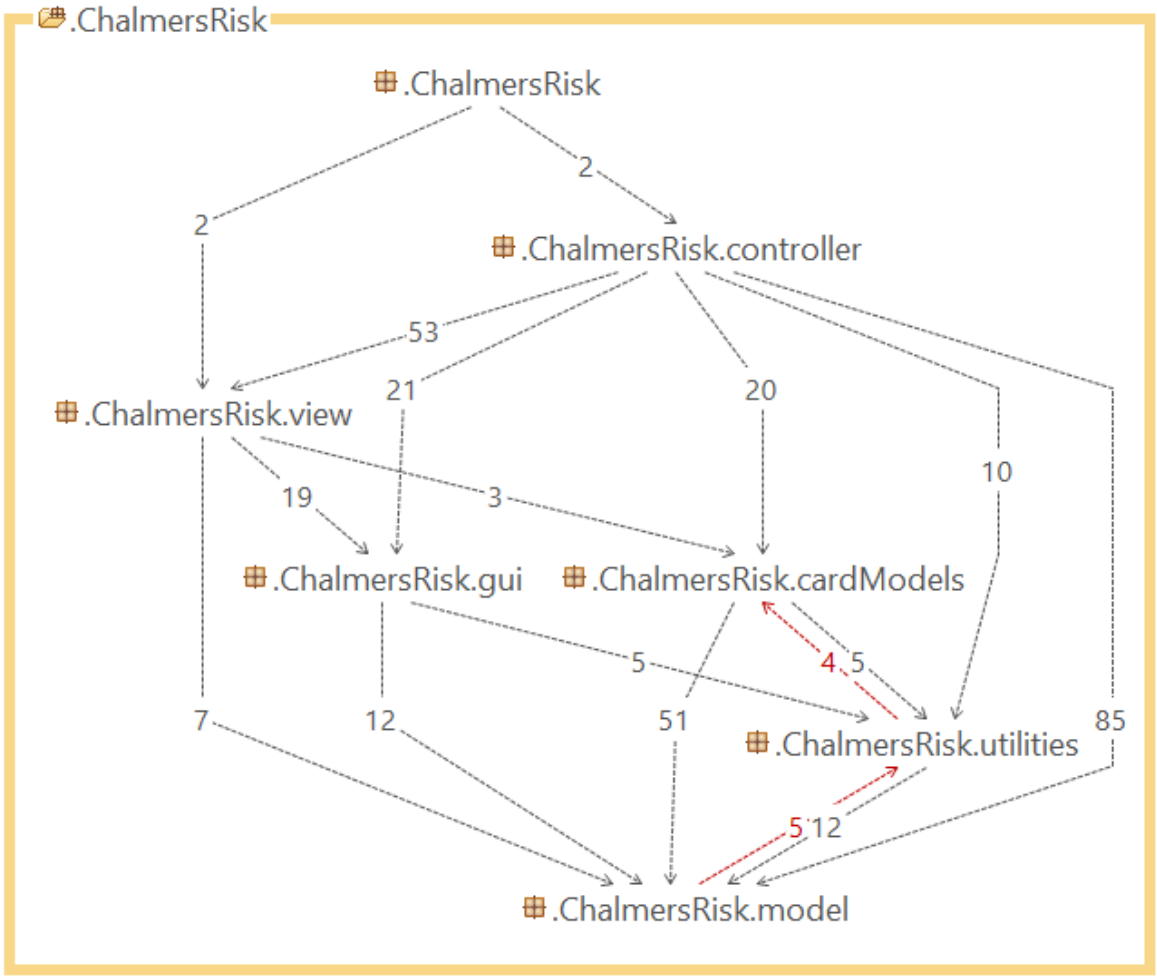


Figure 3 : Layering and dependency analysis

2.3 Concurrency issues

There are no concurrency issues, the game is a single threaded application.

2.4 Persistent data management

NA - The application does not implement any method for saving games between sessions.

2.5 Access control and security

NA

2.6 Boundary conditions

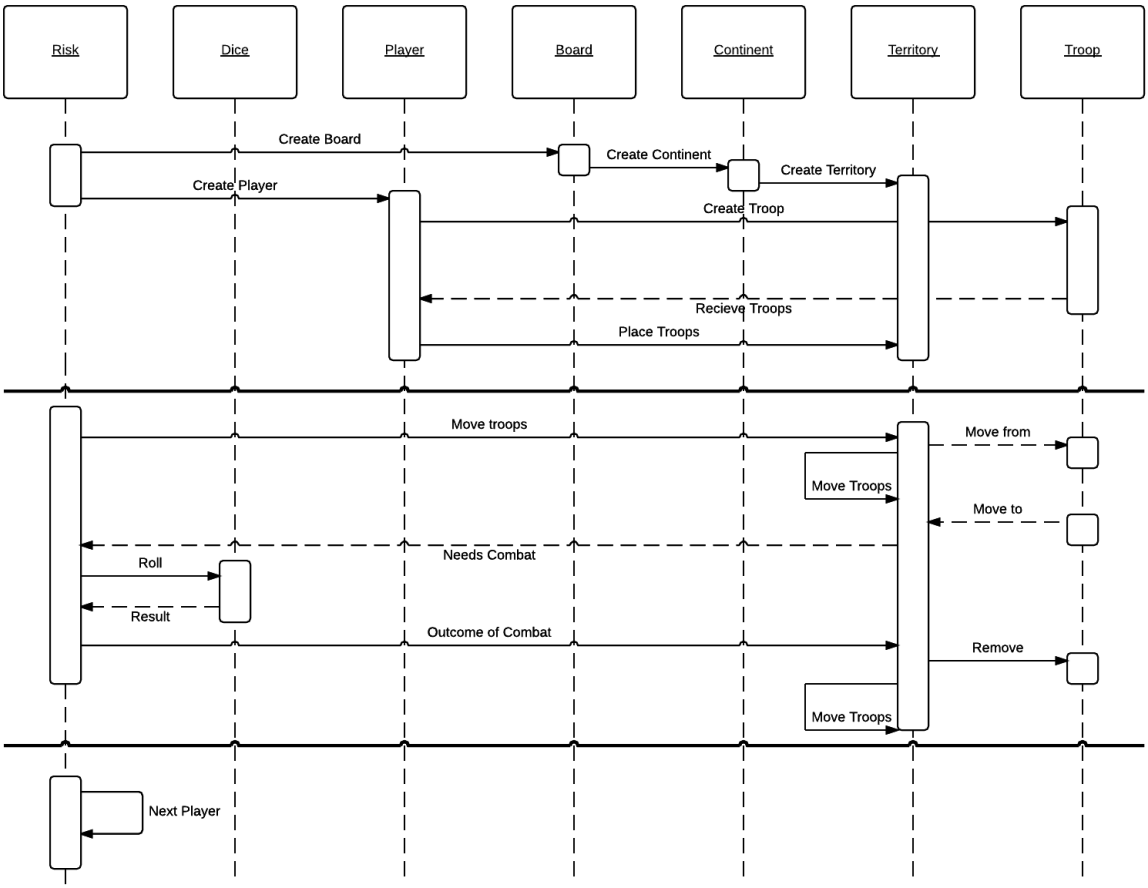
NA - The application is launched and exited as a normal java file.

3 References

MVC: http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
RISK: http://en.wikipedia.org/wiki/Risk_(game)
Fisher-Yates shufle: http://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle
Depth-first search: http://en.wikipedia.org/wiki/Depth-first_search
Bubblesort: http://en.wikipedia.org/wiki/Bubble_sort

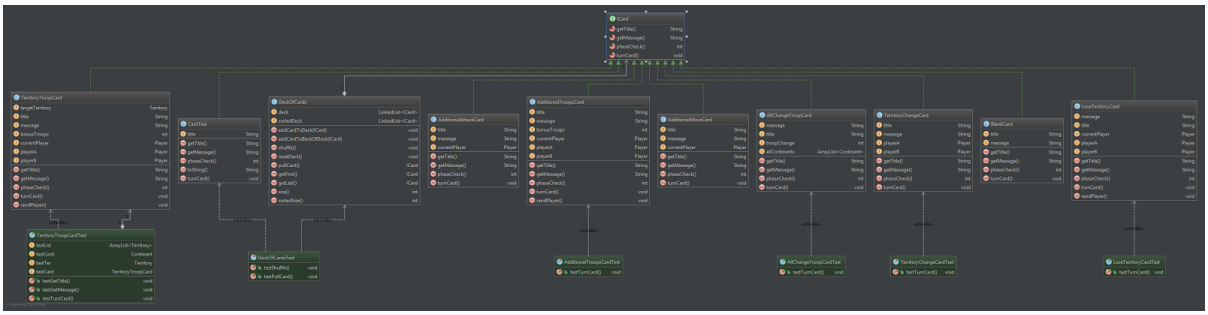
4 Appendix

4.1 Squence Diagram

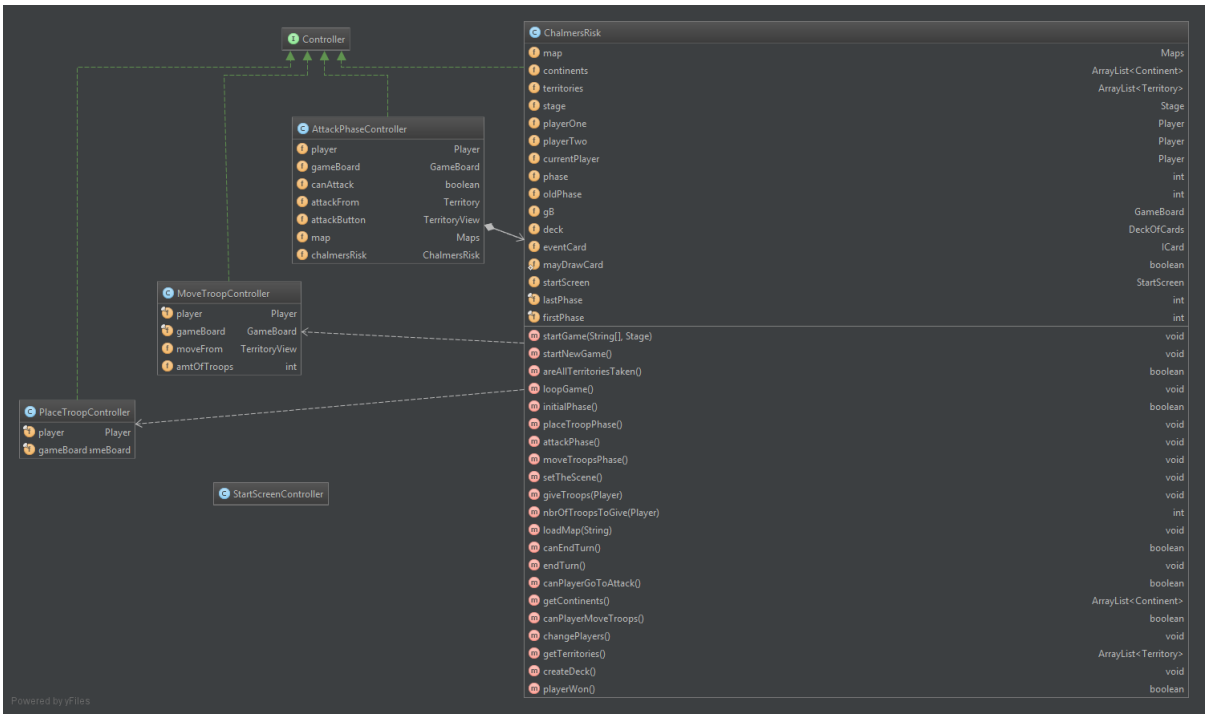


4.2 UML class diagrams

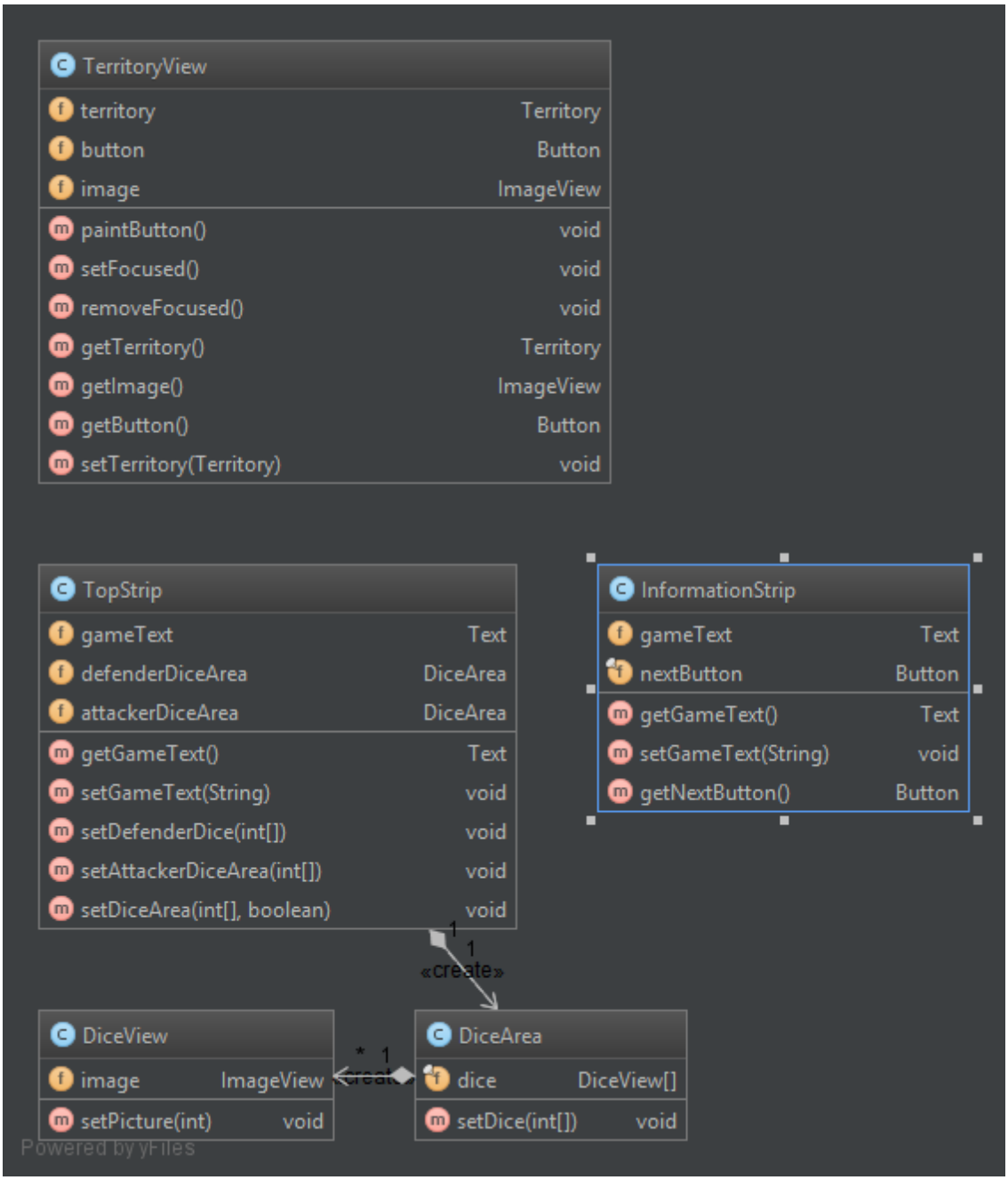
4.2.1 cardModel package



4.2.2 controller package

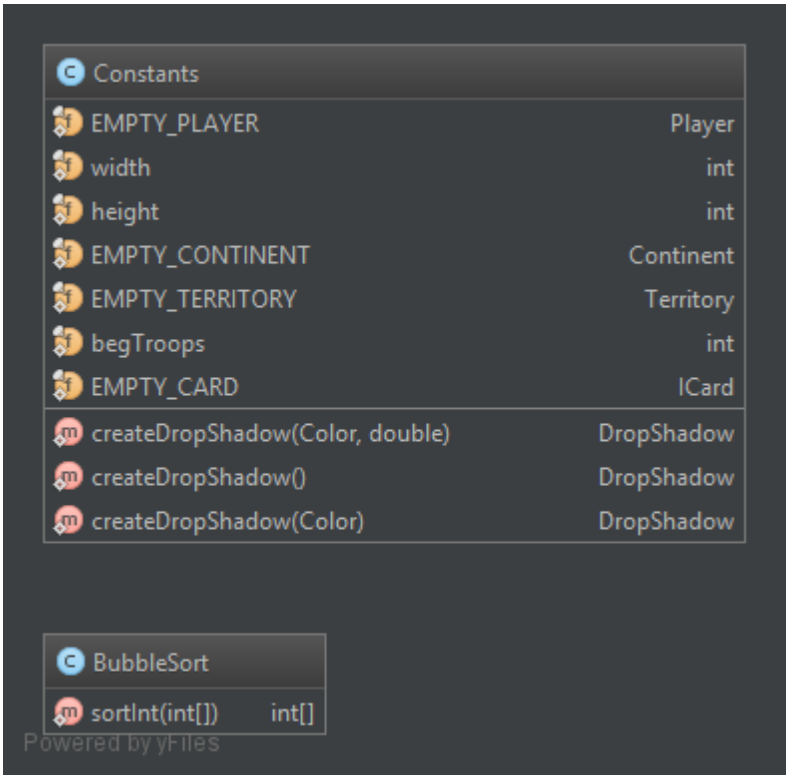


4.2.3 gui package



4.2.4 model package

4.2.5 utilities package



4.2.6 view package