**SOEN 6441: RISKGAME BUILD 2 DOCUMENTATION:**

**ARCHITECTURE DESIGN**

**GROUP MEMBERS:**  
**40087621 An Nguyen**  
**40104630 Naga Satish**

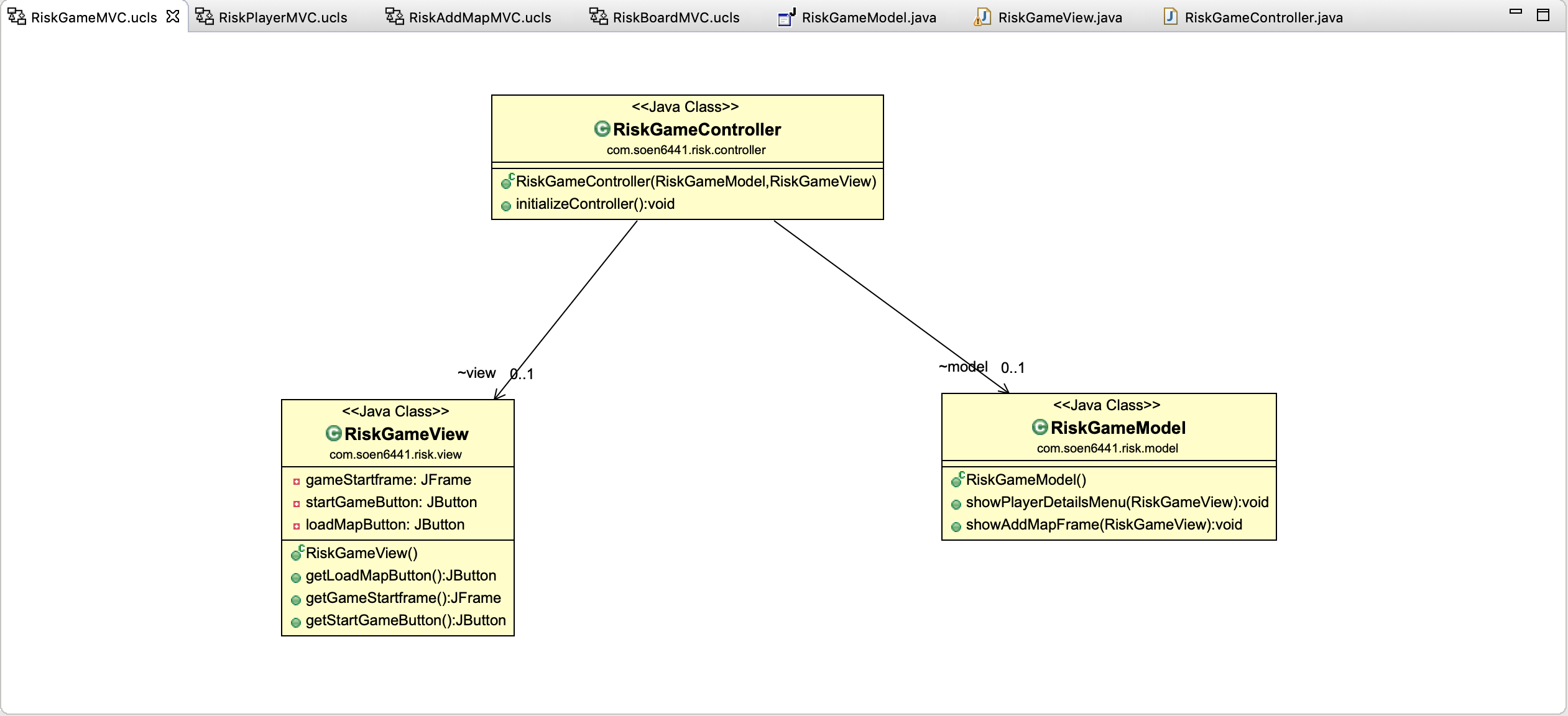
**Scope:**

The Scope of this project is determined by the build requirement uploaded on Module which is an operational model of the popular Risk game.

**Objective**

The Objective of this project is to simulate a real scenario of a project build from design to implementation. Here we will develop the popular risk board game. An initial simplified project requirement has been given which we as team have analyzed.

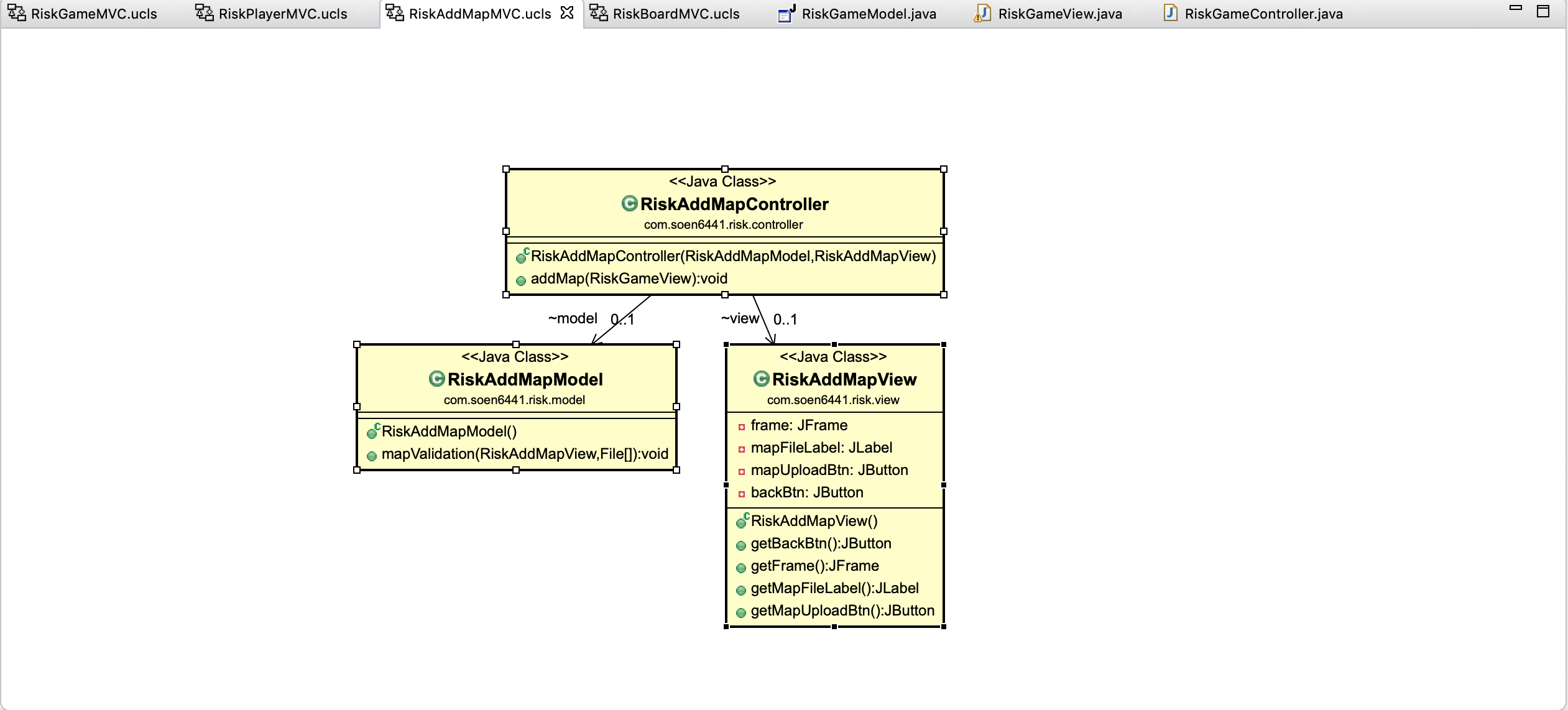
**Architecture Design:**

For this project, MVC pattern has been chosen from build 1 and as build 2 required us to implement the project using observer pattern, MVC is naturally being kept to going forward for build 2 as well. We are using Java frame to display the view the users going to interact with to play the game. Each Model, View and Controller file will be store in different package allocated for each respective file.

**RiskGameView, RiskGameController, RiskGameModel:**

First, we have the RiskGameView class file that display a frame that presents the user with 2 options: Start the game or choose to upload a new map to the system.

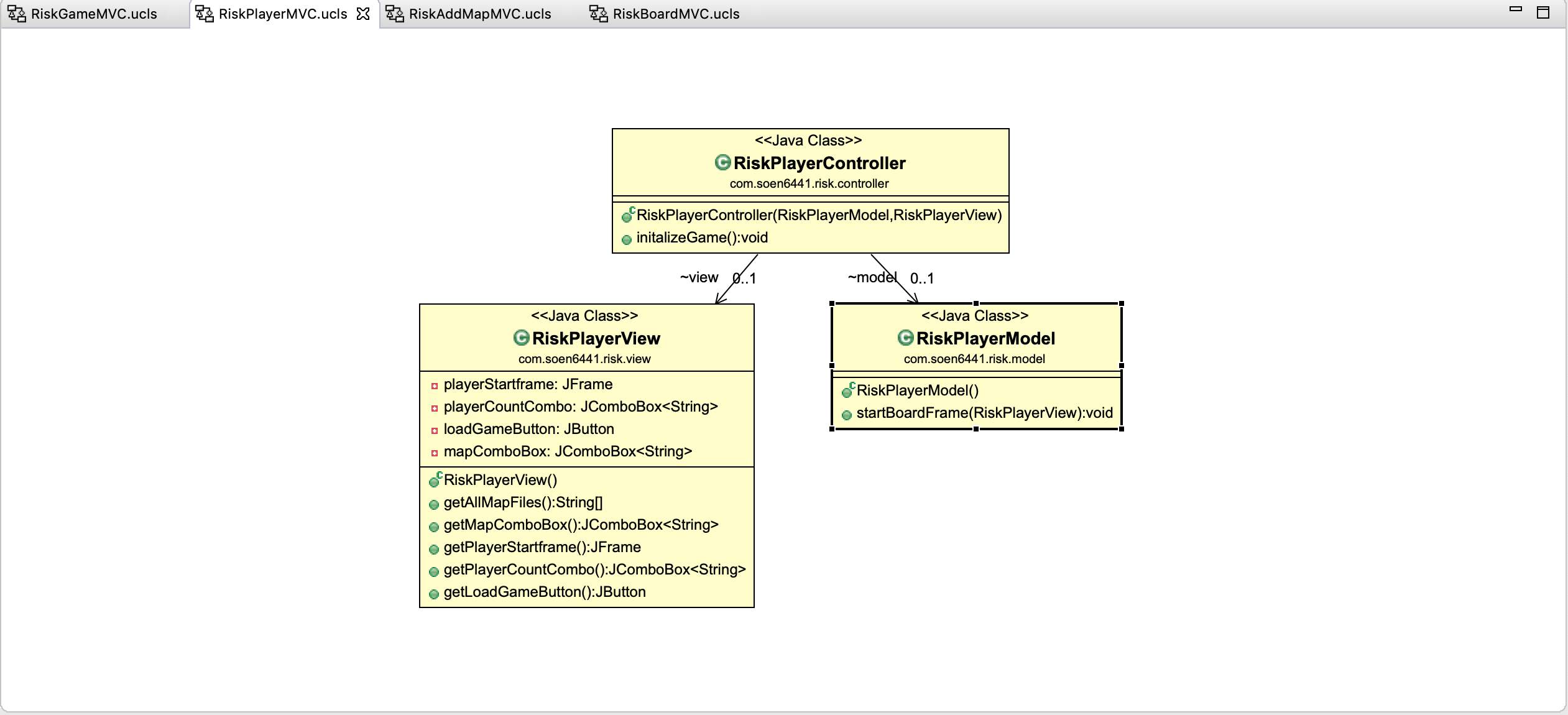
For each button on the frame, an action listener for each button will be defined in the RiskGameController class file and the actual logic for each button activities will be in the RiskGameModel class file. Each button will disable the current frame and enable the next frame according to the user choice.

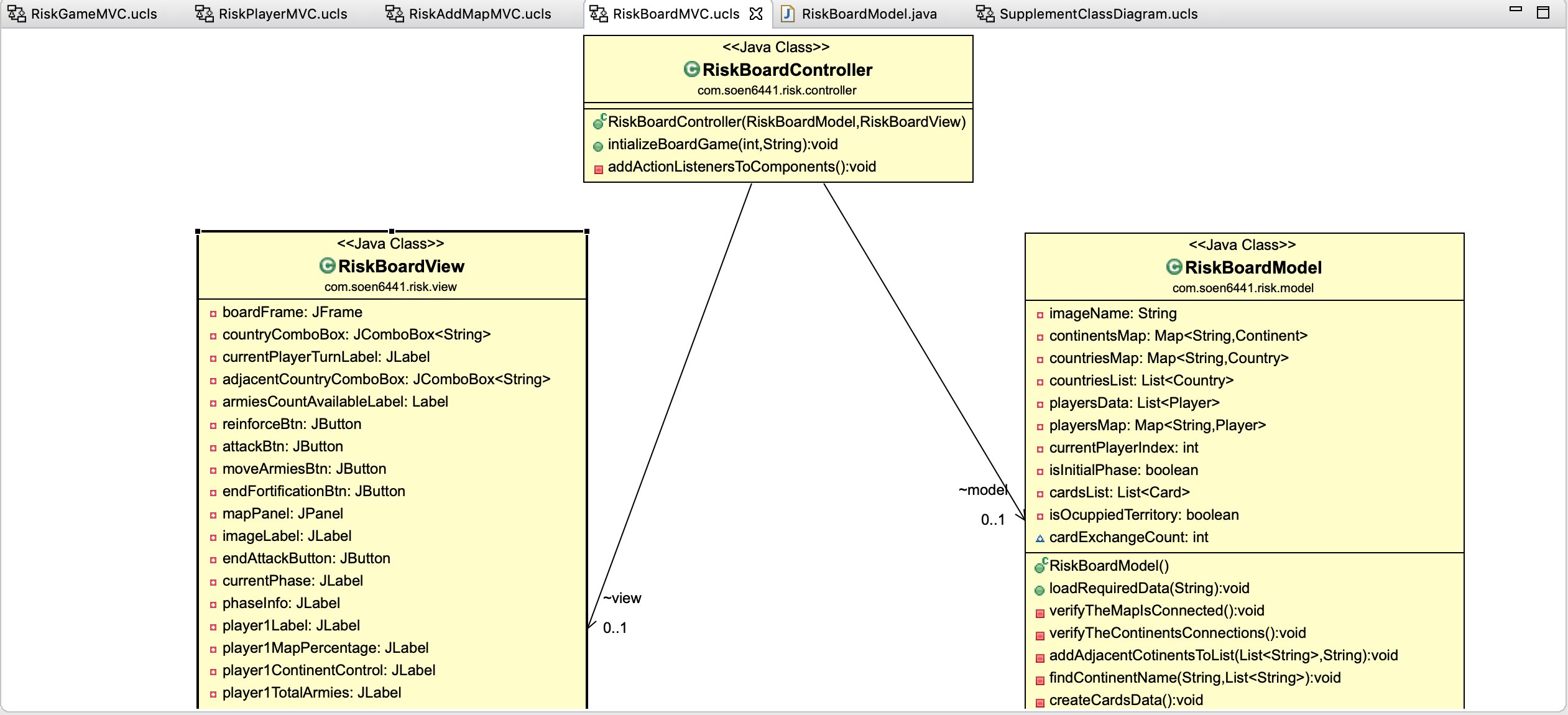
**RiskAddMapView, RiskAddMapController, RiskAddMapModel:**

This frame will allow the user to choose 1 map file and 1 image file of the map to upload and save to the resources folder. This folder stores files of different maps to play the game.

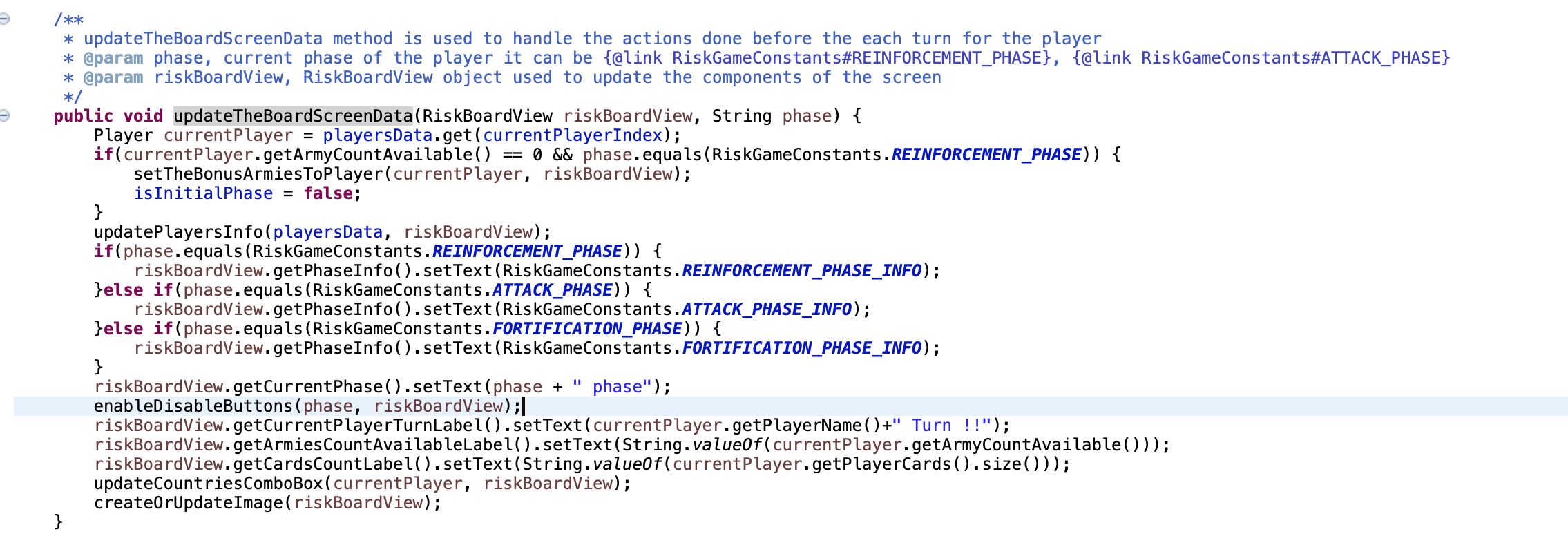
Again, the buttons action listener will have the same structure (Controller file has the action listener for each button and the logic of those buttons will be handle in the model file). If the uploaded map file is incorrect, the program will close. If the map has no problem, it will proceed to the next frame.

**RiskPlayerView, RiskPlayerController, RiskPlayerModel:**

On this frame, the user will be allowed to select a map they want to use playing the game and the number of players (2 to 6 players) the game will be played with. The structure of the Controller and Model file will be the same as the previous MVC files. After finishing choosing, next frame will be the main one for players to play the game.

**RiskBoardView, RiskBoardController, RiskBoardModel:**

RiskBoardView class file contains all of the components of the game and will be displayed on the frame displaying various information such as different buttons that allows the user to choose from for different phase of the game. For example: in Reinforcement phase, the user can place the armies gained at the beginning of the turn on their own countries and if they have sufficient cards, they can choose to exchange cards in this phase.

All of the logic implemented in the game such as attack, fortification and reinforcement are all defined in the RiskBoardModel class file. Information about current player’s turn, which phase is it, all players statistic is also displayed in this frame. To align with the Observer pattern of MVC architecture, a method in RiskBoardModel which is updateTheBoardScreenData is defined to always be called to update the information changed during the course of the game.

Noted that different buttons for different phase will be displayed according to the current phase of the game. Only the buttons corresponding to a specific phase will be displayed.

**Supplementary classes:**

Aside from these classes, different classes have been defined. Those classes are: **Player, Card, Continent, Country, Dice, RiskGameConstants.**

Here are some UML diagram of those classes and the relationship between those:

* Player, Country and Continent class:

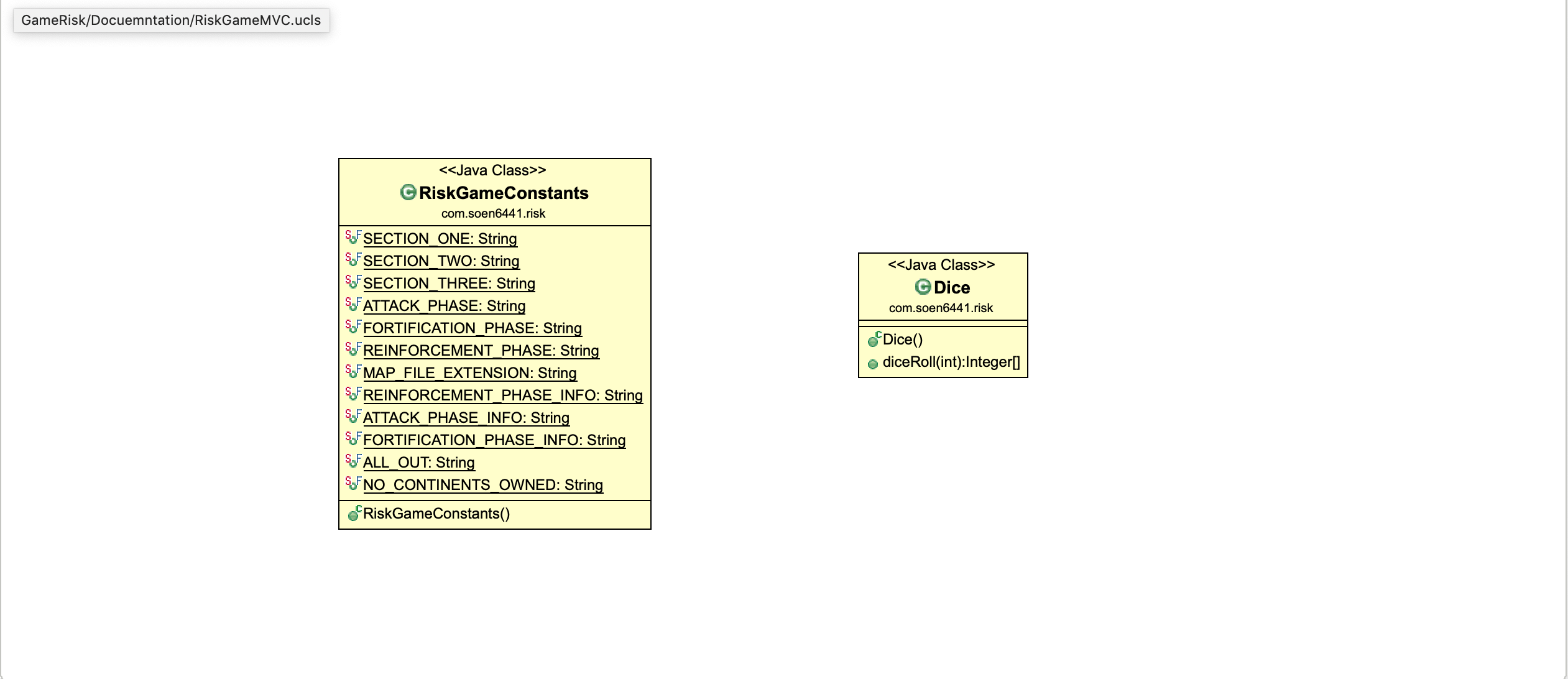
Player class contains information such as player name, list of cards each player owns, list of countries a player currently has. Some refactoring has been applied in that the logic implementation of the reinforcement, attack and fortification have been moved to be included as methods of the player class compared to build 1 in which those methods are implemented as method of the RiskBoardModel class.

Country class contains information for each country of the game. Each country object will have data like x and y coordination that correspond to the image file of the map and how many armies are currently on that country.

Continent class will only have information like name of continent, bonus armies of that continent if the player owns that continent and a list of countries in that continent.

Card class will have information about types of card and in which country that card belongs to. For this project, we decided to have 3 different type of card which is Infantry, Cavalry and Artillery.



* Other classes (RiskGameConstants and Dice class):

Dice class does not contain any information. It has only one method that is diceRoll which is used when a player decides to attack other player’s territories and use it to roll the dices.

RiskGameConstants class is used to store information of constants which are used throughout the coding of the project.

**Junit test classes:**

Une image contenant capture d’écran

Description générée automatiquementCurrently there are 4 Junit test classes which are: PlayerTest, RiskGameModelTest, RiskBoardModelTest and DiceTest.

For RiskGameModelTest and RiskBoardModelTest classes, a test suite class is created to run all of the test cases in those 2 classes. The same has been done for PlayerTest and DiceTest. In each of the test class, a setUpBeforeClass() function has been defined to generate the necessary data for running the test.

Une image contenant capture d’écran

Description générée automatiquement

Une image contenant capture d’écran

Description générée automatiquement

Also, another test suite has been created to run all of the test cases in those 4 test classes.

Une image contenant capture d’écran

Description générée automatiquement