ARCHITECTURAL DESIGN DOCUMENT

AIM OF THE PROJECT:

To build a simple “RISK” computer game that will be compatible with the rules and map files and the command-line play of the “Domination” version of Risk.

ARCHITECTURAL DESIGN MODEL:

Model View Controller (MVC) architectural design

SOFTWARE DEVELOPMENT APPRAOCH:

Extreme Programming approach has been adapted for a straightforward and effortless software development process using its features such as continuous integration, small releases, simple design, testing, etc.

THE MODEL VIEW CONTROLLER ARCHITECTURE:

Model–View–Controller is a software design pattern commonly used for developing user interfaces which distinguishes the related program logic into three interconnected elements. This is done to separate internal representations of information from the ways information is presented to and accepted from the user

The division is as follows:

Model: The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data.

View: The View component is used for all the UI logic of the application.

Controller: Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output.

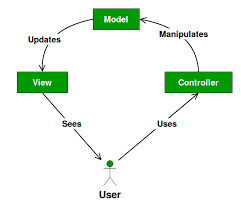


Figure : MVC ARCHITECTURE

Implementation of the MVC idea in our project:

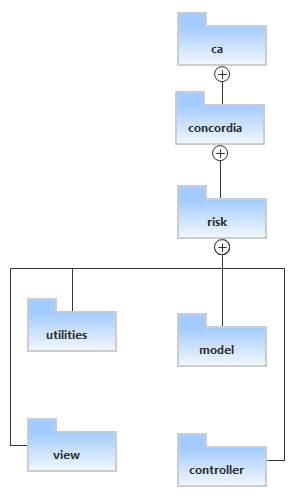
1. **Model** classes in our project namely Map, Country, Continent, Card and Player classes encapsulate the main application logic and domain entity description.
   1. It responds to any state query from the view classes and broadcast notification of state updates if any.
   2. It represents data that is being transferred between controller components or any other related business logic.
   3. It manipulates data and sends back to the database or uses it to render the same data. It responds to the request from the views and also responds to instructions from the controller to update it.
   4. It is also the lowest level of the pattern which is responsible for maintaining data.
2. **View** on the other hand renders the model into a form suitable for visualization or interaction, in a form of UI (user interface).
   1. If the model data changes, the view must update its presentation as needed. Console class manages this responsibility in our project.
   2. Views are created by the data collected from the model data.
   3. A view requests the model to give information so that it resents the output presentation to the user.
3. **Controller** acts as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output.
   1. Controller is also responsible for invoking new views upon conditions. The controller interprets the mouse and keyboard inputs from the user, informing model and the view to change as appropriate. 

Figure MVC for our RISK game

Below Class diagrams are the representation of our Risk game Project with the adaptation MVC architecture:

1. Model Classes in **ca.concordia.risk.model** package

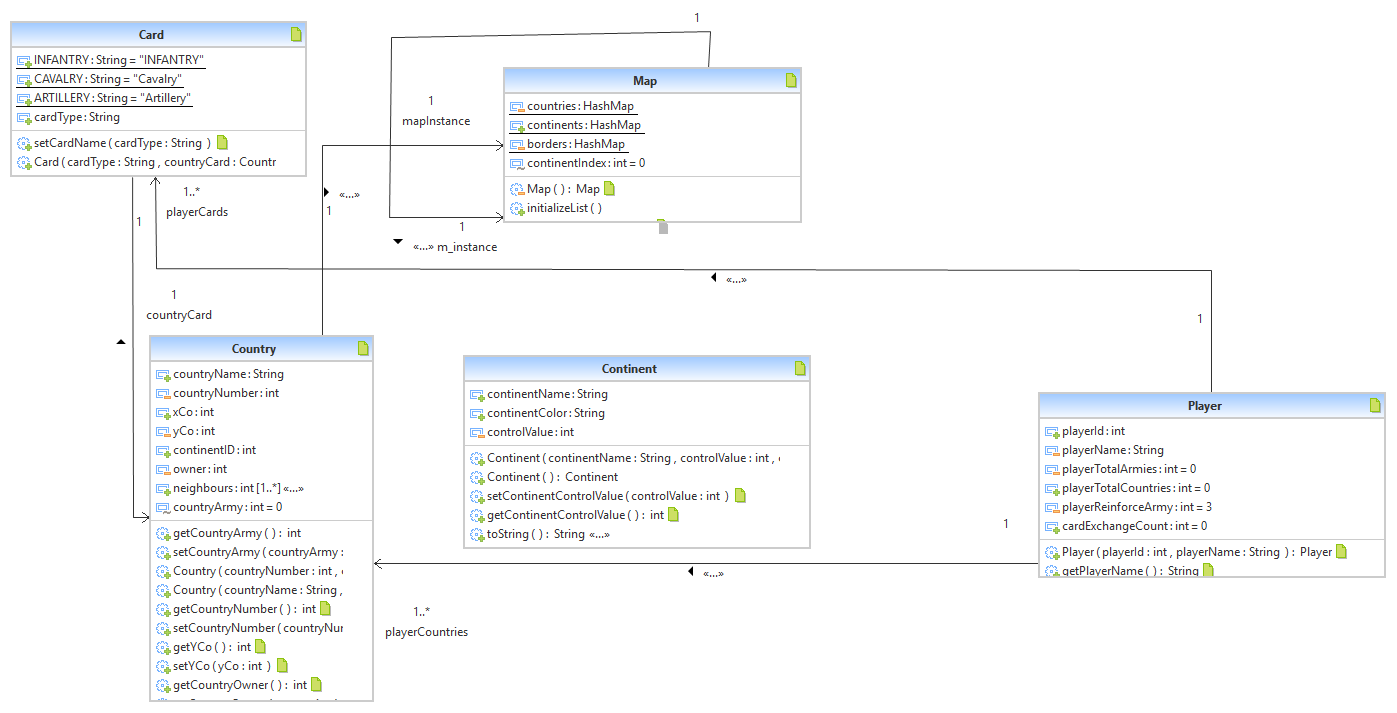


Figure Class diagram for the model classes

1. Controller classes in **ca.concordia.risk.controller** package

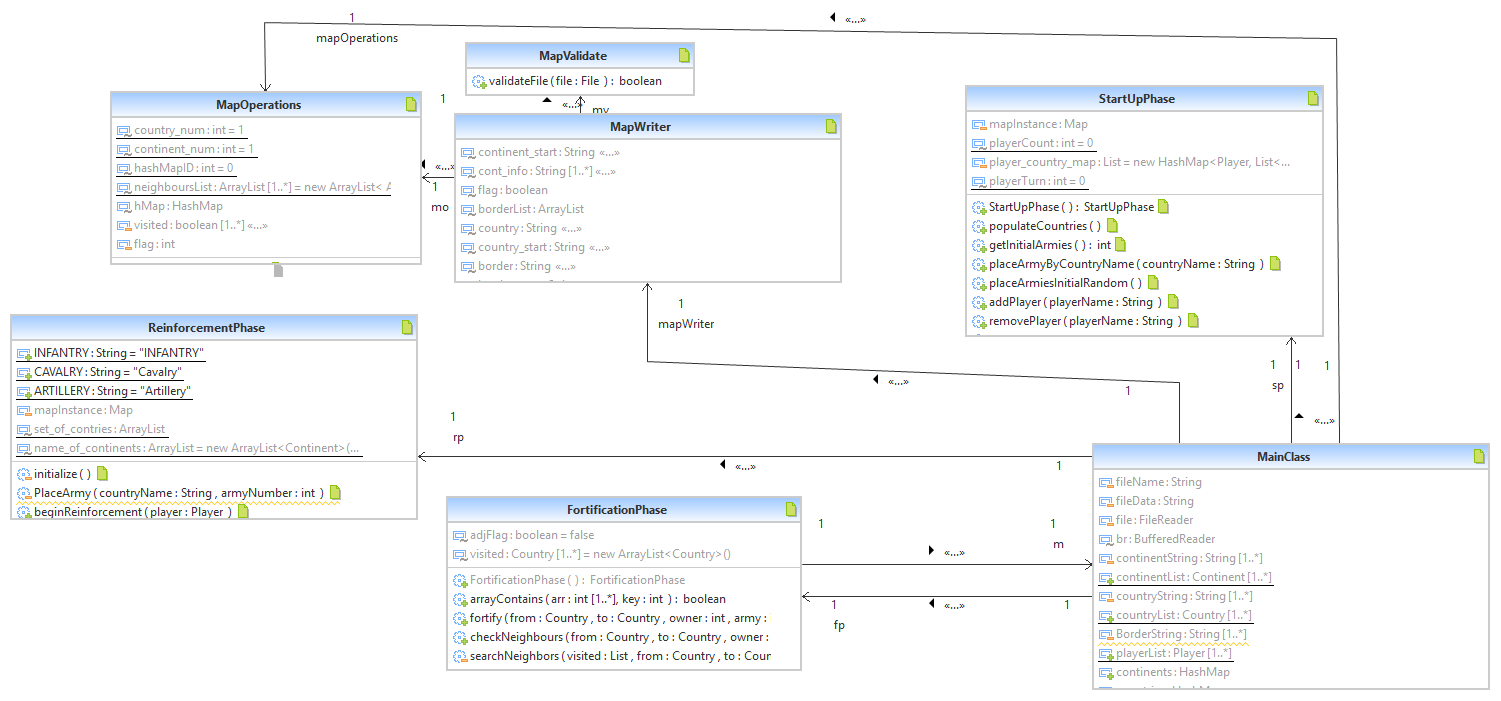


Figure Class diagram for Controller classes

1. Console View in **ca.concordia.risk.view** package and its association with the **controller** MainClass.java

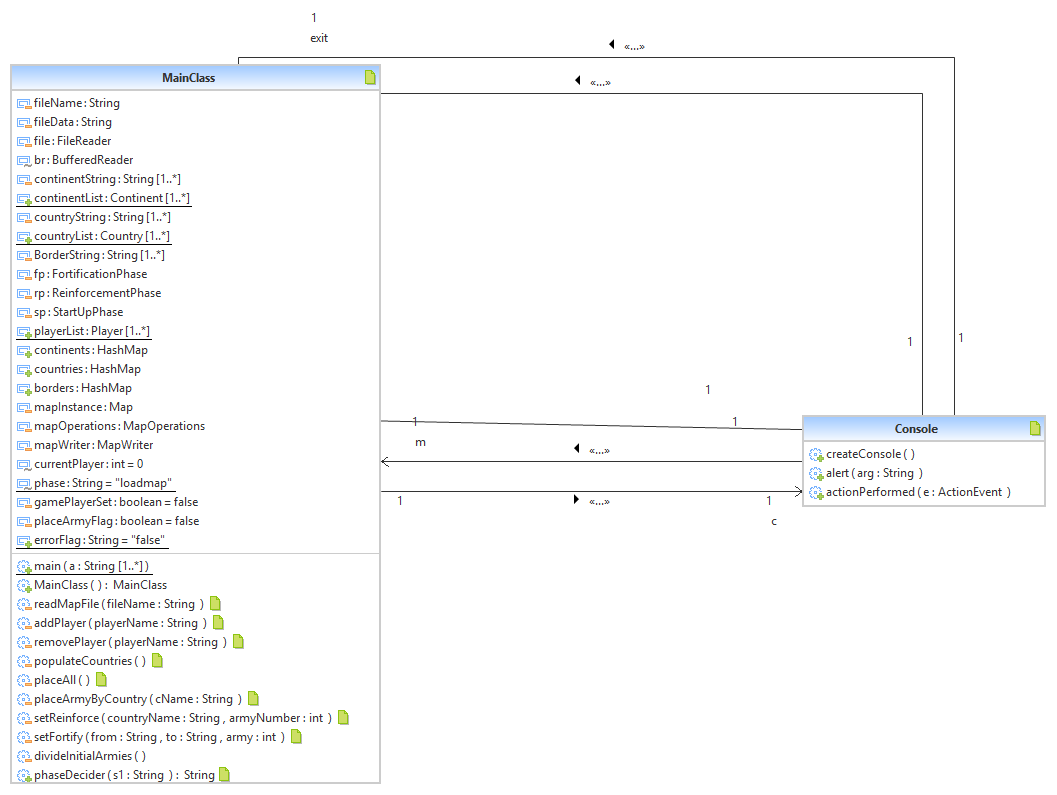


Figure Association of View and Controller