**BUILD 1 ARCHITECTURE DESIGN DOCUMENT**

**INTRODUCTION:**

This is the project’s first build with the main focus on few of the core functionalities such as the map editor and loader. Other functionalities such as the player turn and reinforement have also been implemented. With a build filled with intricate details and multiple functionalities, we felt that an “Incremental approach” would be a better fit for this especially with it’s compatability to extreme programming. Our approach helped us produce set of small working releases that were tested at the unit level before integrating with one another.

As for the design pattern, we have gone forth with the Model-View-Controller. The reason behind being the fact that it allowed us to produce separate modules that are loosely coupled. Hence, comprehension and integration became easy and testing required less amount of time.

Development Model – Agile (Incremental)

Design Pattern – MVC

**MVC ARCHITECTURE**

**Model (col.cs.risk.model)**: Basically defines what data is being used for the application. It is also the one that informs the ‘View’ of any change that has occurred which may or may not be displayed. This change maybe introduced by the ‘Controller’ section.

In case of our RISK Build 1, We have a separate package for the ‘Model’ part of MVC. This provides a more refined view and standard of coding which is followed throughout the build.

* Constants – Contains all our “Constants” that are being ued or resued throughout the code. In order for better understandability and clear-cut differentiation, we have made it available for all classes.
* Continent Model – Stores all of the continent details such as adjacent territories and scores for the armies.
* Game Model – One of the main model calsses whose functionality is a mix of many. It handles most of the application data along with the verification of the map, printing of details and setting up of players.
* Player Model & Territory Model – Handle the player details, territory setting and phases such as fortification.

**View (col.cs.risk.view):** View provides the visual data to be displayed/that is displayed. This depends on the data received from the Model classes or sometimes even controller. The changes will be notified to the view and respective actions are handled.

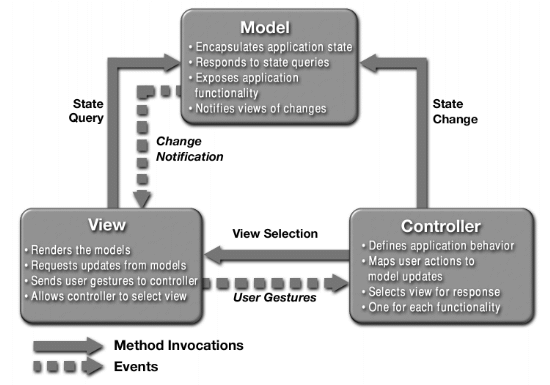
* PlayerSetting – This class is responsible the Jpanel that displays the player selection and the map Selection.
* HopePageViewer – Provides the main GUI for the user to make selections from.
* MapView Classes – 3 View classes that provide the construction of Map from scratch, existing additions/deletions or choosing from the already present.

**Controller (col.cs.risk.controller):** Controller handles the requests from the user which maybe through the GUI(in our case) by providing the respective model data which in turnn reflects on the UI again.

* StartGameController – Is the main class that is responsible for setting up of the UI and fetching and setting of few initial paramaneters.
* GameController – Does the validation of data for the fields such as the continents and territories.
* MapPanel – Has the control functions for calling the displayMap() and does the ‘connecting of nodes’ to display the connected graph.
* PlayerPanel – Handles the player data such as the number of armies received and also color-code the players to avoid confusion.

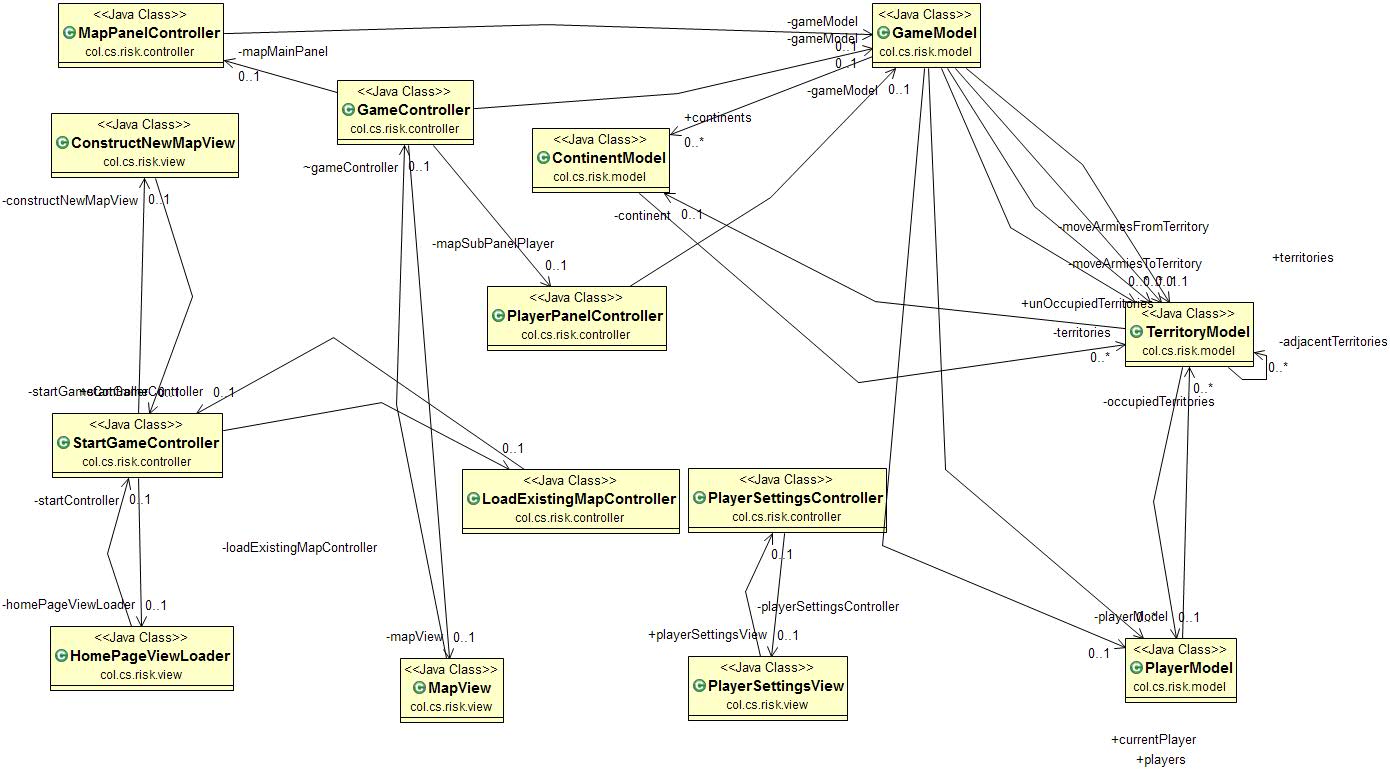
**Extreme Programming Features Implemented**:

1. Planning : A general plan for the not-so-distant future and a more precise plan for the current work was built that laid the basic foundation for the five of us to work on.
2. Collective Ownership : Everyone’s the boss of oneself. All the resources and workload were shared amongst everyone.
3. Sustainable Pace: From day 1 of the build to this date, All of us had made it a point to meet up, discuss, provide insights and much more ensuring that no one had felt extra pressure or any sort of discomfort.
4. Continuous Integration : With a distributed Repository such as the GIT hosting our code, we were able to test and integrate our error free code frequently. (Commits)
5. Small Releases : On a regualr basis, we were able to develop and deploy a partially working applicaton with certain set of incremental features for every release.



1. Represents the MVC architecture being used

**UML DIAGRAMS:**



1. Class Diagram

References: