**BUILD 2 ARCHITECTURE DESIGN DOCUMENT – TEAM 25**

**INTRODUCTION:**

This is the project’s second build with the main focus on few of the core functionalities such as the attack phase and card exchange. Other functionalities such as the observer pattern and dice roll functionality 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

**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.

**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.

* CardExchangeModel – Observable class that provides the changes made in the state of the card exchange.
* CardModel – Model Class for assigning the card and to return the type of card assigned.

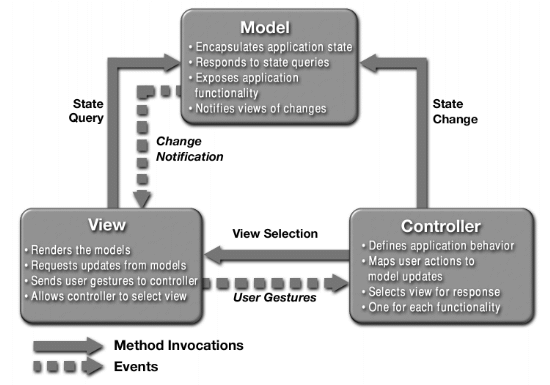
**View (col.cs.risk.view):** View provides the visual data to be displayed/that is displayed. This depends on the data received from the Model classes or sometimes even controller. The changes are then displayed as an UI to the user.

* MapView – Addition of the Card button to provide an

**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.

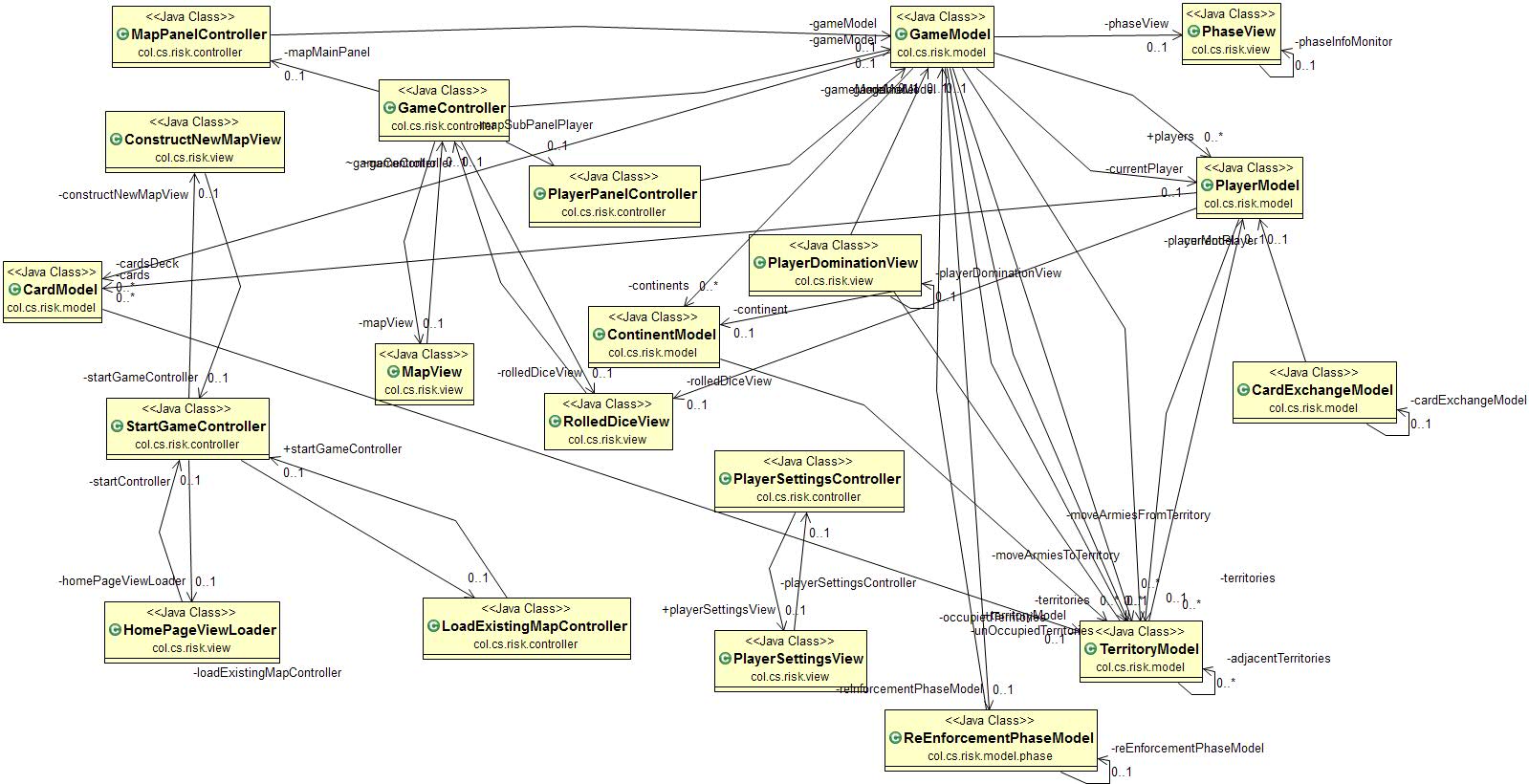
**Phase (col.cs.risk.model.phase):** Provides a bundle of observable classes and an interface that generalizes the functions needed for the observable classes.

* **AttackPhaseModel** – Notfies the observers of the changes that occur during the attack phase of each player, every turn.
* **EndPhaseModel** – Handles the case when the ‘End of a Phase’ has reached.
* **FortificationPhaseModel** – Similar to the AttackPhaseModel, provides the observers of the details of the players during the fortification phase.
* **ReinforcementPhaseModel** – Provides the players details during the reinforcement phase.
* **StartPhaseModel** – Botifies the basic player details such as the current player and the total number of players currently playing the game.
* **GamePhase** – An interface for generalizing a set of functions for the Phases.



1. Represents the MVC architecture being used

**UML DIAGRAMS:**



1. Class Diagram



1. Package Diagram

References:

1. <https://tallyfy.com/uml-diagram/>
2. <http://www.extremeprogramming.org/>
3. <https://users.encs.concordia.ca/~paquet/wiki/index.php?title=SOEN6441_-_fall_2018>