SOEN 6441- Advanced Programming Practices

Winter 2019

Architectural Design Document for Risk Game Project

Submitted to: Prof Amin Ranj Bar

Group 11

Gursimran Singh -	40080981
Karandeep Karandeep -	40104845
Namita Faujdar -	40105179
Karanbir Singh Pannu -	40047216
Shriyans Athalye -	40037637

Contents:

1. Introduction

1.1 Purpose of the Document

This documents purpose is for building an architecture design model implementing predefined architectural model. The implemented design model was Model View Controller also following some of the known programming approach for smooth and effective game development.

- **Programming Concepts**: Some of the well-known programming practices were followed to maintain effectiveness of project development. Example such as pair programming where two programmers worked on same workstation to maintain productivity.
- **Simple Design**: Simple workflow of design made it easier to read and avoid faults.
- **Continuous Integration**: Bitbucket was used as version control for the project where all programmers were made to work with single branch and commit accordingly. Maintenance of frequent changes, rollbacks helped increase the productivity by automatic integration.
- **Coding standards**: Simple and understandable coding standards were followed including naming and file organization conventions.

1.2 Scope of Document

This design covers the development of simple risk game using different builds covering different phases of game. Detailed explanations of design are covered in different sections below. The motives of design decisions made were crucial for implementing entire project working and help effective build. This document will cover overall project architecture by explaining from the base until final build process. Covers all the aspects of Build1 according to requirement. This document covering the representation of the following of MVC design pattern for implementing user interfaces on computers. It divides a given application into three interconnected parts. This is done to separate internal representations of information from the way's information is presented to, and accepted from, the user. The MVC design pattern decouples these major components allowing for efficient code reuse and parallel development

The scope of the build 1 is as per the instruction guidelines for the build:

MapEditor: Covering the following below functionality, which is a form of a connected graph with proper interconnection between the continent and territories and abiding to the conquest map file format.

- ➢ Create a new map file
- ➢ Edit an existing map file
- ➢ Add/update/delete Continent
- ➢ Add/update/delete Country
- ➢ Add/Delete Adjacent country
- ➢ Make sure that the integrity of the connected graph is maintained.

Game Play: The game play covers:

- ➢ Assigning country to player
- ➢ Player ability to assigning armies to each country in round robin manner
- ➢ Reinforcement phase, with proper calculation of armies ➢ Fortification phase, with a valid fortification move.

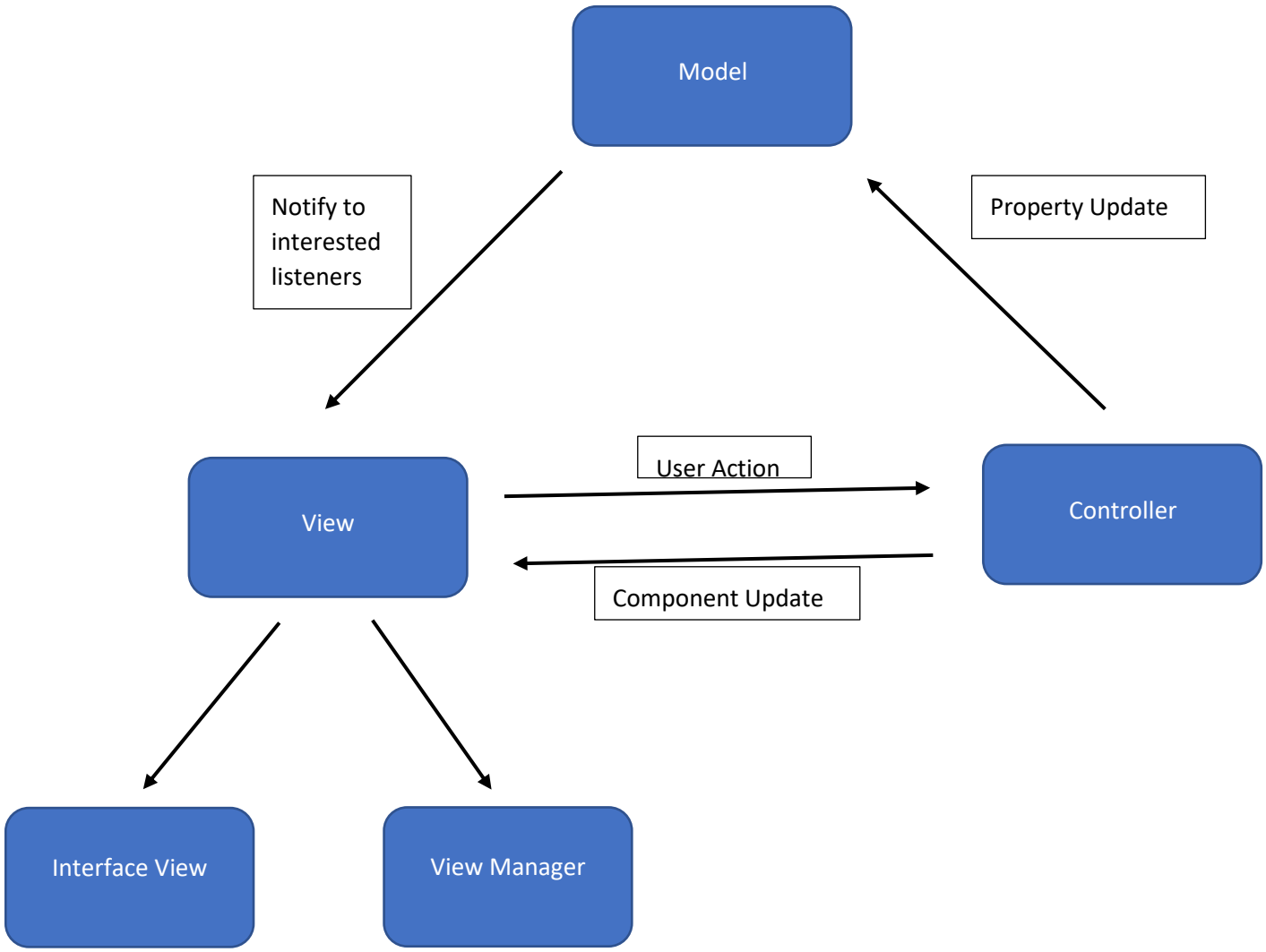
2. MVC Architecture

Model view controller helps to decouple data access and business logic from the way it is displayed to the end user. This distribution of logic is done in different sections of the MVC architecture namely model, view and controller.

Model: Model being the lowest level of this architecture, it represents data and the rule that govern access to and updates of this data. Model mainly is responsible for maintaining data.

View: View renders contents of a model. View specifies how any model's data should be presented. If incase any model data changes, view responsibility is to update its presentation as needed. The update feature can be achieved via a push model where view registers itself with model for change notifications or pull model for calling model whenever it retrieves most current data.

Controller: Controller translates user's interaction with view into actions that model will perform. If considering standalone application, user interactions could be click of button or mouse event. Depending on the context controller may also select new view like web page of result for representation purpose to the user.



In Development use:

- **Model**: This model manages data of application domain. Main models of our project consist of **Country model, Continent Model, GamePlayModel, PlayerModel** where the application domain remains. If model gets a request for change from View, then they respond to instructions via controller.

- **View**: In View, it renders model into a form suitable for visualization or interaction in a form of UI. If model data changes, view must update its presentation as needed. Some of the views in our project are **MainGameview, MapEditView, StartupView, FortificationView**. This will be notified by model accordingly.

- **Controller**: Controller is designed to handle user input and initiate response based on event making calls on appropriate model objects. This will instruct model to perform operations by accepting user inputs. For example**, StartupController** is used to call the model to perform operation.

- **Interface View:** Interface View extends observer to maintain state change. This interface consist of methods show view, hide view, show message.

- **View Manager:** View Manager is designed to handle different interface such as MainGameView, MapCreateContinentView, MapCountryConnectView. This will handle from single file rather from controlling from different file.

3. Technology and Tools:

3.1 **Technologies and tools used for the development of the game**:

Technology and Tools	Description
IntelliJ	IDE for the game development
Maven	Maven as a build automation tool to manage all project dependencies.
JavaFX	Library to control the UI components of the Risk Game
Junit 4	Junit 4 for writing test cases