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

Index:

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:

Functional requirements:

Map Editor:

- ● User-driven creation of map elements, such as country, continent, and connectivity between countries.
- ● Saving a map to a file exactly as edited (using the "conquest" game map format). Loading a map from an existing "conquest" map file, then editing the map, or create a new map from scratch.
- ● Successfully load the 3dCliff.map and World.map file. Reject loading of map file by correctly verifying that continents are connected subgraphs

Single game mode:

● Game starts by user selection of a user-saved map file, then loads the map as a connected graph. User chooses the number and behavior of players (see Player Behavior Strategies below). The game proceeds until one of the players has conquered the whole map. If no human player is selected, the game proceeds fully automatically without any user interaction.

Start-up Phase:

● All countries are randomly assigned to players. Players are allocated a number of initial armies, depending on the number of players. In round-robin fashion, the players place one by one their given armies on their own countries.

Reinforcement phase:

● Calculation of correct number of reinforcement armies according to the Risk rules. Players place reinforcement armies on the map. Reinforcement ends automatically when all armies have been placed. Implementation of a "card exchange view" using the Observer pattern.

Fortification phase:

● Implementation of a valid fortification move according to the Risk rules. Fortification ends automatically when the armies have been 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.

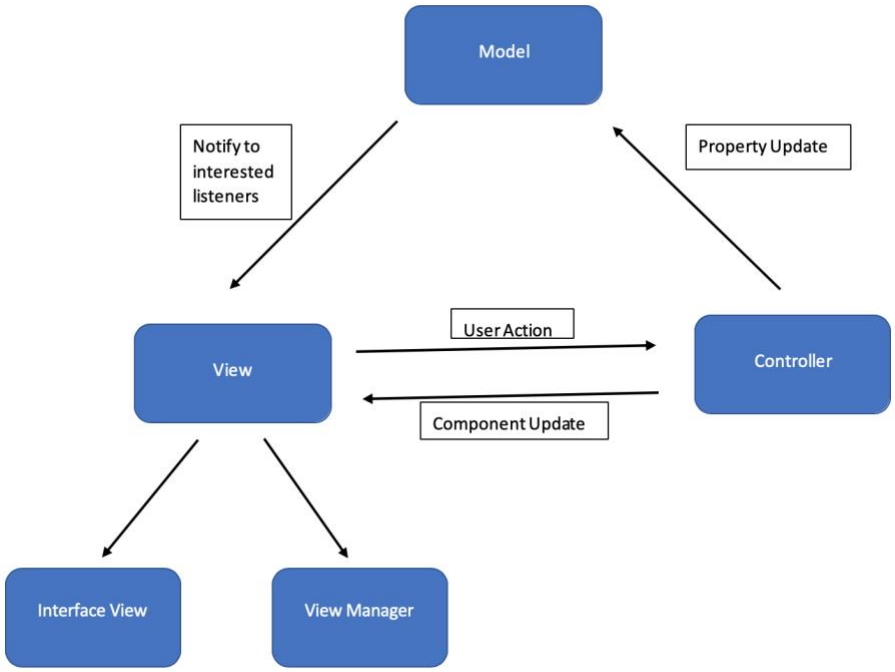

Figure 1 Model View Controller

Development use:

- **Model**: This model manages data of application domain. Main models of our project consist of **Country model, Continent Model, MapRIskModel, 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 AWT**MainGameview, AWTMapEditView, AWTStartupView, AWTFortificationView**. 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 AWTMainGameView, AWTMapCreateContinentView, AWTMapCountryConnectView. This will handle from single file rather from controlling from different file.

Modules Description

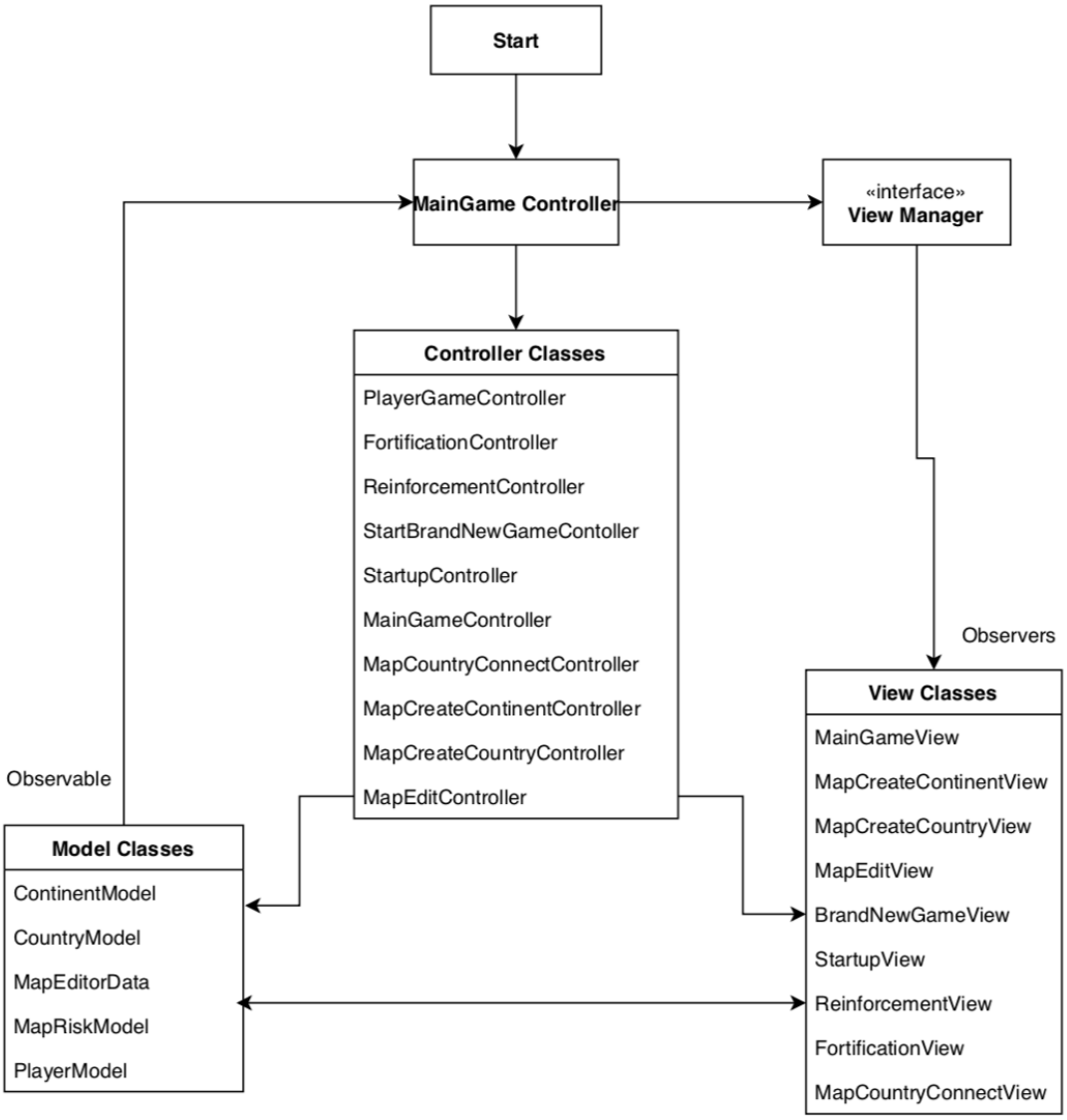

Figure 2 Architecture Diagram

VIEW Classes

AWTMainGameView	Creates a welcome screen. This View has the UI components for the main game view.
AWTMapCreateContinent	Add continents with their control values to map file.
AWTMapCreateCountry	Add countries to particular continents. It adds countries to the map while creation.
AWTMapEditView	The edit control value of continents and link between countries and continents.
AWTBrandNewGameView	Here we browse and upload the map to start the game.
AWTStartUpView	Startup phase of the game. Map loaded an initial army assignment to players.
AWTReinforcement View	Calculate a number of armies and assign to players.
AWTFortification View	Move armies from one country to another.
AWTMapCountryConnectView	View for players to connect a country to another country during map creation

CONTROLLER Classes

Fortification Controller	The Fortification Controller handles the data change between model and view respectively and updates with the changes detected on either side.
PlayerGame Controller	The player game controller mainly handles the data change and update among the game risk model and view,
Reinforcement Controller	Reinforcement controller takes care of the movement of the data from model corresponding to the view, and updates whenever there is change detected.
StartBrandNewGameController	This controller is used to control the data when a new game is started.
StartUpController	Allocate armies to respective player.
MainGameController	Starting point of game.
MapCountryConnectController	The MapCountryConnectController is used
MapCreateContinentController	The CreateContinentController controls and updates the model and view when a continent is created in the map editor.

MapCreateCountryController	The CreateCountryController controls and updates the model and view when a country is created in the map editor.
MapEditController	It does the movement of data into the model corresponding to the view and controller. It also takes care of updating view whenever data is changed or updated.

Model Classes

Continent Model	Stores all continents with the control value.
Country Model	Stores all countries linked to continents.
MapEditor Data	Observable list of countries and continents.
MapRisk Model	Helps in creating and editing a new map.
PlayerModel	Stores data of all players.

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 and Swing	Library to control the UI components of the Risk Game
Junit 4	Junit 4 for writing test cases

Bibliography

[1] "Risk Game Rules," [Online]. Available: https://en.wikipedia.org/wiki/Risk_(game).

[2] "MVC Architecture," Oracle, 2019. [Online]. Available: https://www.oracle.com/technetwork/articles/javase/index-142890.html.

[3] "Junit Tutorial," [Online]. Available: https://www.tutorialspoint.com/junit/.