Department of Engineering and Computer Science

Concordia University

Montreal, Quebec, Canada

Advanced Programming Practices

(SOEN-6441)

(Winter 2019)

Architecture Design

Project: Risk: Strategy Build-3

Team-35

Name	Student ID
Gursharan Deep	40039988
Harmanpreet Singh	40059358
Raghav Sharda	40053703
Yaswanth Choudary Narra	40087595
Saman Soltani	40093581

Architecture Design

Architecture:

Project has been implemented using **Model-View-Controller (MVC)** architecture.

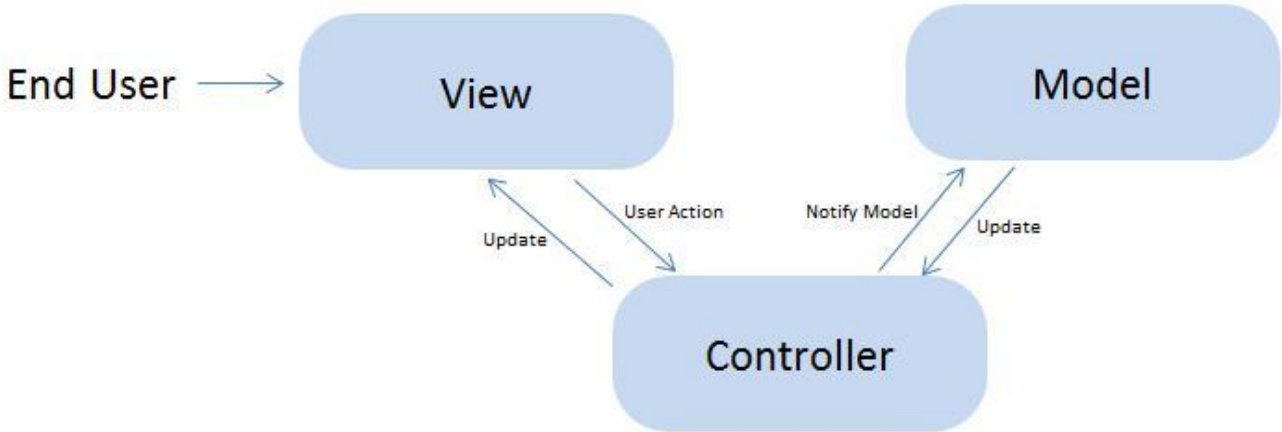
MVC Architecture:

- Model View Controller architecture secludes the application logic from the user interface layer and facilitates separation of tasks.
- It is a 3-tier architecture model and is widely used for many object-oriented designs with user interaction.
- This architecture has three components
    - Model
    - View
    - Controller

- **Model:** It is the deepest level of pattern which is responsible for maintaining the data.

- **View:** It is responsible for displaying data to the user.

- **Controller:** It handles the interaction between Model and View.



SCOPE:

The scope of this project incorporates the following requirements:

- **Build Requirement**: Deliver an operational version demonstrating a subset of the capacity of the system. This is about demonstrating that the code build is effectively aimed at solving specific project problems or completely implementing specific system features.

- **Functional Requirements:**

    - MapEditor:
        - User-driven creation of map elements, such as country, continent, and connectivity between countries.
        - Saving a map to a file exactly as edited.
        - Loading, editing, or creating a new map from the scratch.
        - Verification of map correctness upon loading and before saving.

    - Startup Phase:
        - Game starts by user selection of a user-saved map file, then loads the map as a connected graph. User chooses the number of players, then 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.

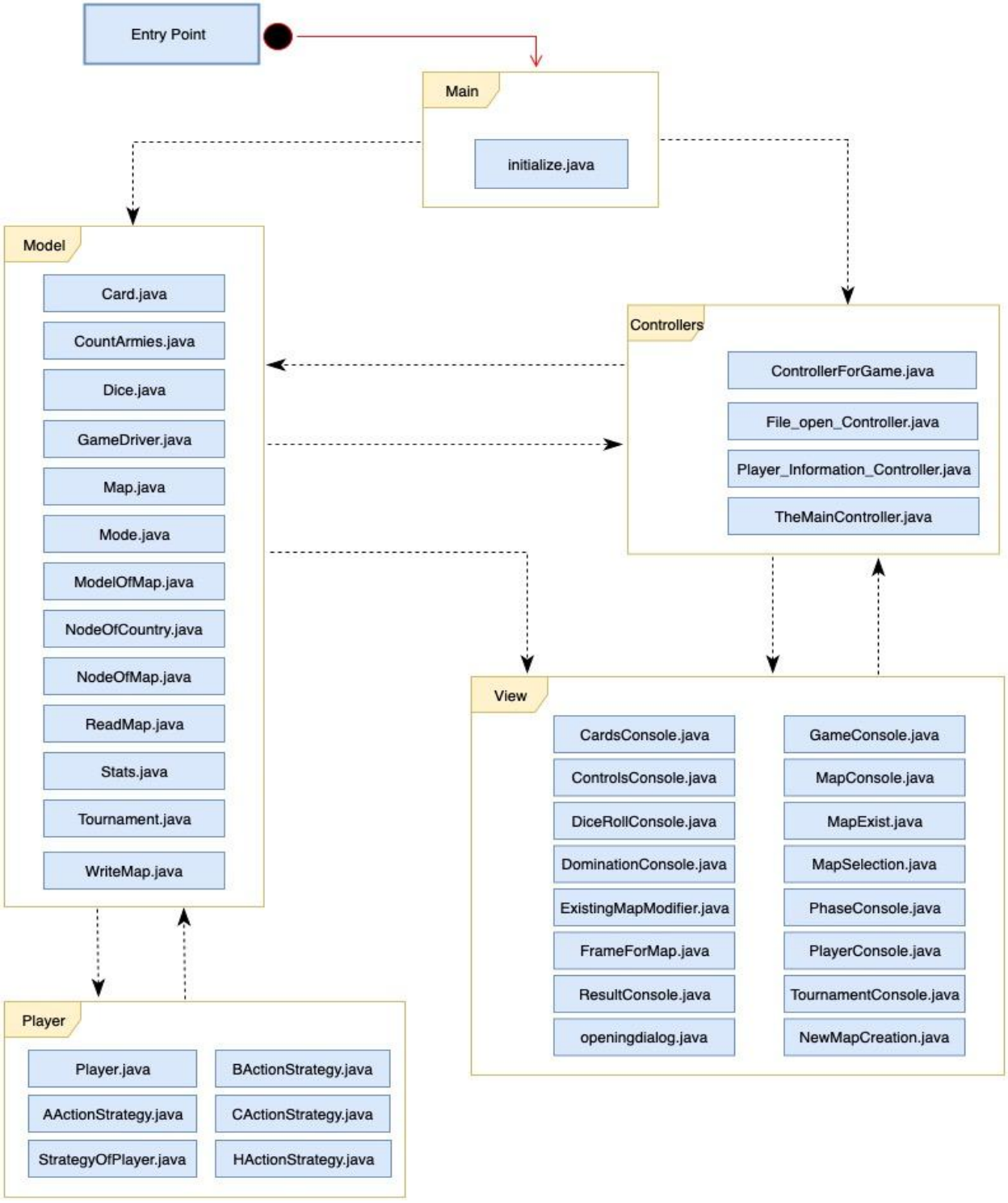
    - Game Play:
        - Implementation of a “phase view” using the Observer pattern
        - Implementation of a “players world domination view” using the Observer pattern.
        - Implementation of the reinforcement, attack and fortification as methods of the Player class.
        - Implementation of Tournament Mode which proceeds without any user interaction and display results at end of this mode.

- Reinforcement Phase:
    - Calculation of correct number of reinforcement armies according to the Risk rules.
    - Implementation of a “card exchange view” using the Observer pattern. The card exchange view should be created only during the reinforcement phase.
    - The cards exchange view should cease to exist after the cards exchange.

- Attack Phase:
    - Player can declare an attack by selecting attacker and attacked country.
    - Attacker and attacked player decide how many dice to roll.
    - Proper number of armies are deducted from attacker/defender country during the attack(s).
    - If defender is conquered, attacker can move any number of its armies in the conquered country (see the Risk rules). If it results in conquering the whole map, the attacker is declared the winner and the game ends.
    - Player may decide to attack or not to attack again. If attack not possible, attack automatically ends.
    - Implementation of an “all-out” mode, where once the attacker and attacked country have been identified, the attack proceeds with maximum number of dice and end only when either the attacker conquers the attacked, or the attacker cannot attack anymore

- Fortification Phase:
    - Implementation of a valid fortification move according to the Risk rules.

- Game Save/Load:
    - Implementation to save the game progress in a file and user can load that save game from that saved file.


Entry Point
Main
initialize.java
Model
Card.java
CountArmies.java
Dice.java
GameDriver.java
Map.java
Mode.java
ModelOfMap.java
NodeOfCountry.java
NodeOfMap.java
ReadMap.java
Stats.java
Tournament.java
WriteMap.java
Controllers
ControllerForGame.java
File_open_Controller.java
Player_Information_Controller.java
TheMainController.java
View
CardsConsole.java
ControlsConsole.java
DiceRollConsole.java
DominationConsole.java
ExistingMapModifier.java
FrameForMap.java
ResultConsole.java
openingdialog.java
GameConsole.java
MapConsole.java
MapExist.java
MapSelection.java
PhaseConsole.java
PlayerConsole.java
TournamentConsole.java
NewMapCreation.java
Player
Player.java
AActionStrategy.java
StrategyOfPlayer.java
BActionStrategy.java
CActionStrategy.java
HActionStrategy.java

Architecture Description of Project:

The whole application of risk game (build-1) is divided into three interrelated layers hiding internal representation from user interaction with the system. It allows code reuse. Following is description of main modules in risk Game.

- **MODEL:** This component of the build determines the behaviour of the game based on the user interaction with the view. Different parts handle state of different game entities like card information, game phases, maps, player information etc. explained below:

  - ArmyCount: Initialises number of armies based on number of players.
  - Card : Contains information of assigning and trading cards between players.
  - Dice : Performs dice operations(Validation and Assigning random values to dice rolls, Assigning Territories and move Armies when a player wins.
  - GameDriver: Handles data of player information, maps and player setup during gameplay.
  - GameTurnDriver : Handles the phases and turn of players.
  - GamePhase : Used for Attack, Reinforcement and Fortification phases of the game.
  - Map : Deals with all data related to conquest maps used in game implemented as **observable** for views. Handles map data and map Editor.
  - NodeOfCountry : Handles data of a country.
  - NodeOfMap : Handles data of a map.
  - ReadMap : Responsible for reading map file from user defined location.
  - WriteMap : Responsible for writing a map to user defined location.
  - Player Package: Is a part of Model Package.
    - HActionStrategy: implements strategy of player
    - AActionStrategy: implements aggressive strategy of player
    - BActionStrategy: implements benevolent strategy of player
    - CActionStrategy: implements cheater strategy of player
    - Player: This class is responsible to represent the player

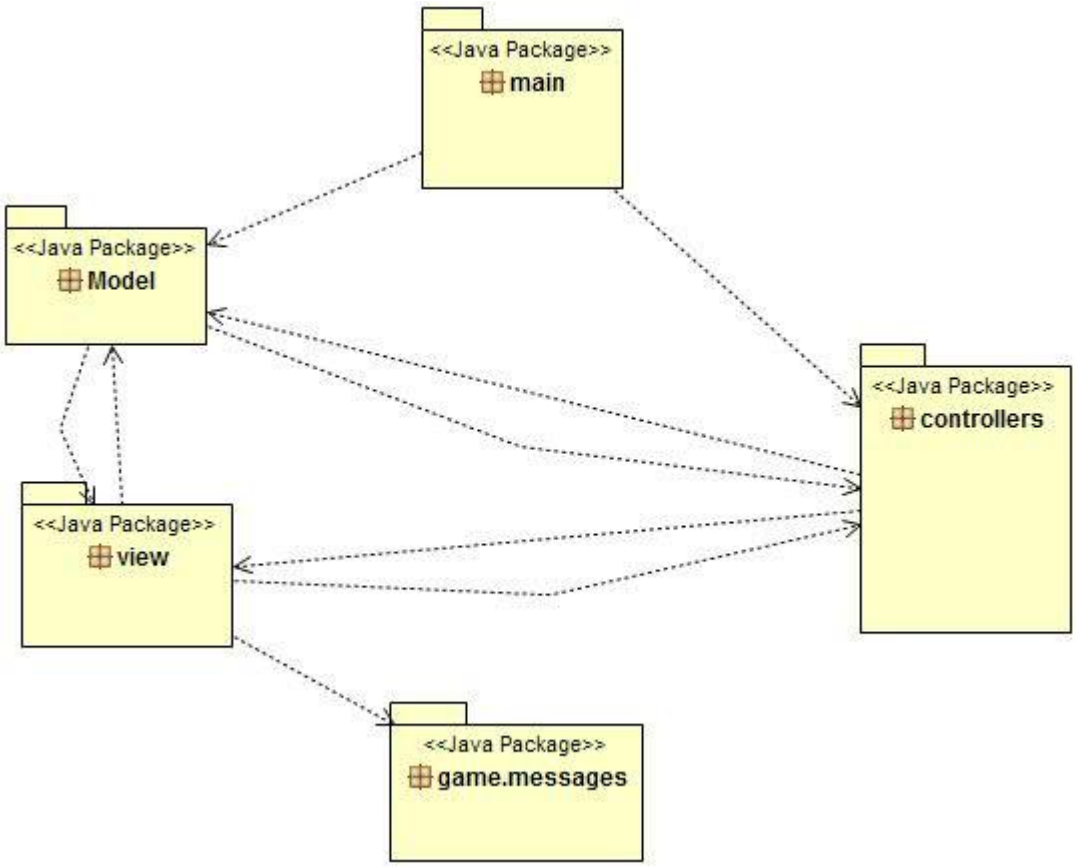
- **VIEW:** It deals with different user interface platform allowing user interaction. There are multiple views in the project and all of them are implemented as **an Observer pattern:**

  - CardsConsole: this class creates the cards view on main game console.

- ControlsConsole: this class creates the view of various game controls like army count etc. on main game console.
- DiceRollConsole:This class creates the dice roll view on main game window
- DominationConsole:This class will implement the domination by different players in the world.
- ExistingMapModifier: This class opens the JFrame view for delete/add country and continent while modifying Map.
- GameConsole: This class is the **main view** of the front end of the game console where the players will play the game and all other information while playing the game will be displayed.
- Map_Frame: opens JFrame view for selecting either a New map or an Existing map.
- MapConsole: set coordinates for map.
- MapExist:This class opens JFrame to open map file and initialise contents of frame.
- MapSelection: It implements view for map file selection
- NewMapCreation: NewMapCreation class opens the JFrame view for creating new map, add/delete country and continent
- Openingdialog: It is main View of the game that opens up when application is initialised.
- PhaseConsole:Implements the Phase View panel of the main window using Observer Pattern.
- PlayerConsole: this class creates the view for Player information display.
- TournamentModeMenu: This class provides a menu to go for tournament mode by providing the info asked.

CONTROLLERS: These classes issue directions to Model classes to update themselves as per user interaction. It handles main game-logic and coordinate communication between VIEW and MODEL. It contains following modules:

- the_main_controller: controller class which controls the main functioning of the game in the initial phase.
- Edit_create_map_controller: controller class that perform actions for creation of new map and modification of existing map. Responds to requests of NewMapCreation and ExistingMapModifier.
- File_open_controller: control the opening of various files from local system during game.
- Player_information_controller: This class will return all the information required for a player-the number of players and name of players



Package Diagram