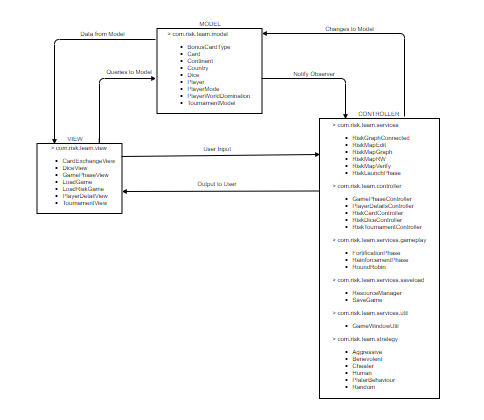
**Architectural Design:**

Our RISK game is designed based on MVC, Observer and Strategy Design Pattern. We have divided and distributed our modules in the parts **MODEL, VIEW, CONTROLLER and SERVICES**. We have different states of game such as Player, Card, Dice in Model Folder. Also View contains all FXML Loader and In Services, we have business logic which is related to Controller in MVC. All FXML layouts are made using JavaFX scene Builder 2.0 application which has the ability to generate the backend coding by itself.



**Modules in Model:**

We have 9 models described below:

1. Player: Player class is used to model the actual player entities of the game and implemented reinforcement, attack and fortification phase.

2. Country: Country class models countryentity, where each player owns few countries in the

beginning and their aim is to get ownership of all the countries.

3. Continent: Continent class models Continent entity, which can be considered as a superset of countries, where each country belongs toone particular continent.

4. BounsCardType(Interface): It is used to model cards which are allocated after the attack phase if any player is eligible for the card.

5. Card: Card class is used to model cards, which are assigned to players after they win a country.

6. Dice: Class to model dice objects for the attack phase.

7.IPlayerType(Interface): Interface to define type of a player.

8. PlayerWorldDomination: This class models the data to be displayed for the player world

domination view.

9. TournamentModel: Class to model the Tournament, it has methods to update Map and

implements logic for plating the game in the tournament

**Modules in Services:**

Our business login is implemented by services, controller and strategy folders. Implements the

game flow, players strategies etc.

1. RiskMapEdit: This class provides functionality to edit or create a map from scratch.
2. RiskMapGraph: RiskMapGraph creates a graph (connected) from map data and provides methods tomodify the graph.
3. RiskGraphConnected: Checks whether a graph is connected or not.
4. RiskMapRW: This class reads the data from a .map file and provides it to RiskMapEdit’s object. Italso gets data from RiskMapEdit’s object and writes it to an empty .map file.
5. RiskMapVerify: It is responsible for verifying the correctness of a map which is tobe loadedfor game play.
6. RiskLaunchPhase: It takes data from RiskMapRW, and initializes data for Players, countries andarmies. Here countries are randomly assigned to each player.

* **Gameplay: It has the following classes.**

1. ReinforcementPhase: It has methods to get the number of armies calculated for are

assignment to each player.

1. FortificationPhase: It provides method to pass the armies form one country to

another.

1. RoundRobin: It provides functionality for round robin traversal among the players.

* **Controller: It has controllers for controlling the models.**

1. RiskCardController: It governs the card view. It loads the card for the game play.
2. RiskDiceController: It governs the attack view. It manipulates the attack view and

updates the dice objects according to the attack phase.

1. GamePhaseController: It governs the game play.
2. PlayerDetailsController: It takes the number of players and, player names.
3. RiskTournamentController: Class to interact with the TournamentView FXML. It takes

inputs from the user and calls Tournament model for operating the Tournament.

* GameUpdateWindow: It has various helper functions related for handling JavaFX features.
* **Strategies :**

1. Aggressive: It governs aggressive players game phases.
2. Benevolent: It governs benevolent players game phases.
3. Cheater: It governs cheater players game phases.
4. Random: It governs random players game phases.
5. PlayerBehavior: Provides an abstract class for player strategies.
6. Human: A Human player that requires user interaction to make decisions.

* **Saveload:**

1. ResourceManager: It provides logic for loading and saving the game.
2. SaveData: To make data serializable for writing object name and its data and to retrive it while reading that saved game file back.

**Modules in View:**

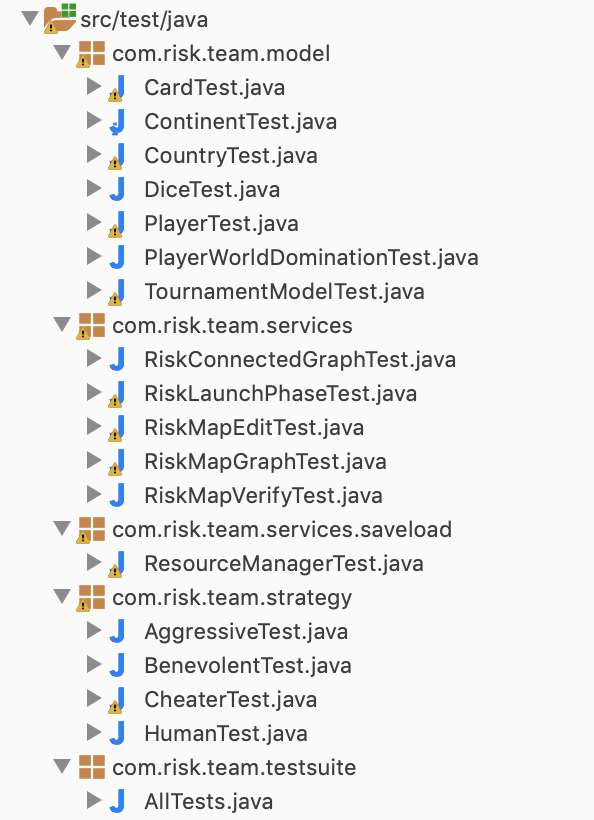
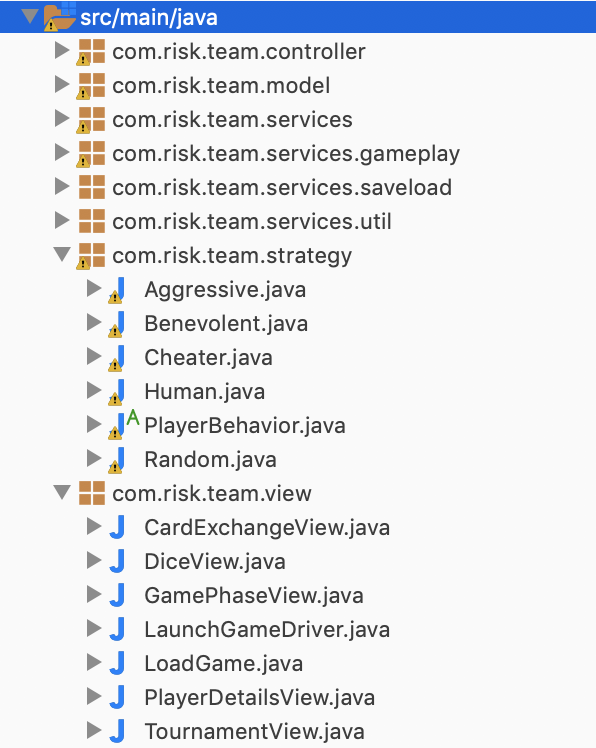
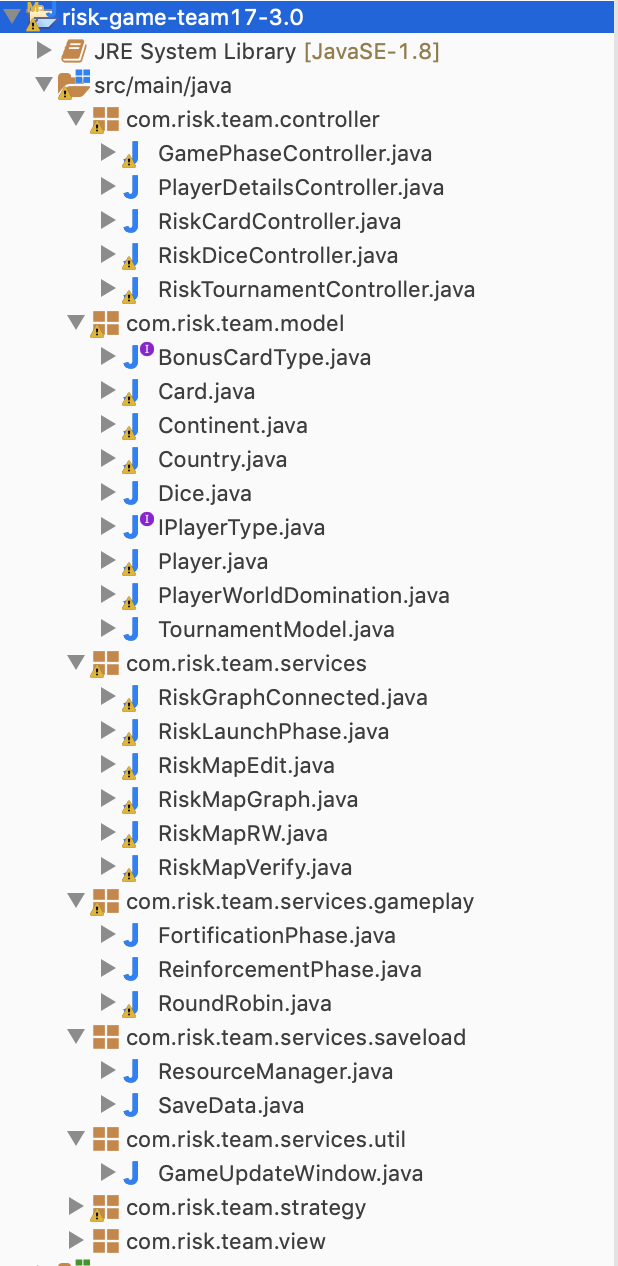
1. DiceView: This view provides a display window for the attack phase. Player interacts with

this window during the attack phase.

1. GamePhaseView: It is used to initialize the game play screen.
2. PlayerDetailView: View for the player selection. It provides options to select number of

players and their names.

1. TournamentView: It basically has loads the tournament view FXML.
2. CardExchangeView: Load card view FXML before reinforcement.
3. LoadGame: Loads MapLayOutSelector.FXML for loading an already saved game.
4. LoadRiskGame: This class loads LoadGame file to start game window.



Same hierarchy is followed for **Test Folder** and **TestClasses**.There is one to one Mapping of Each Java Tested Class to Test Class.