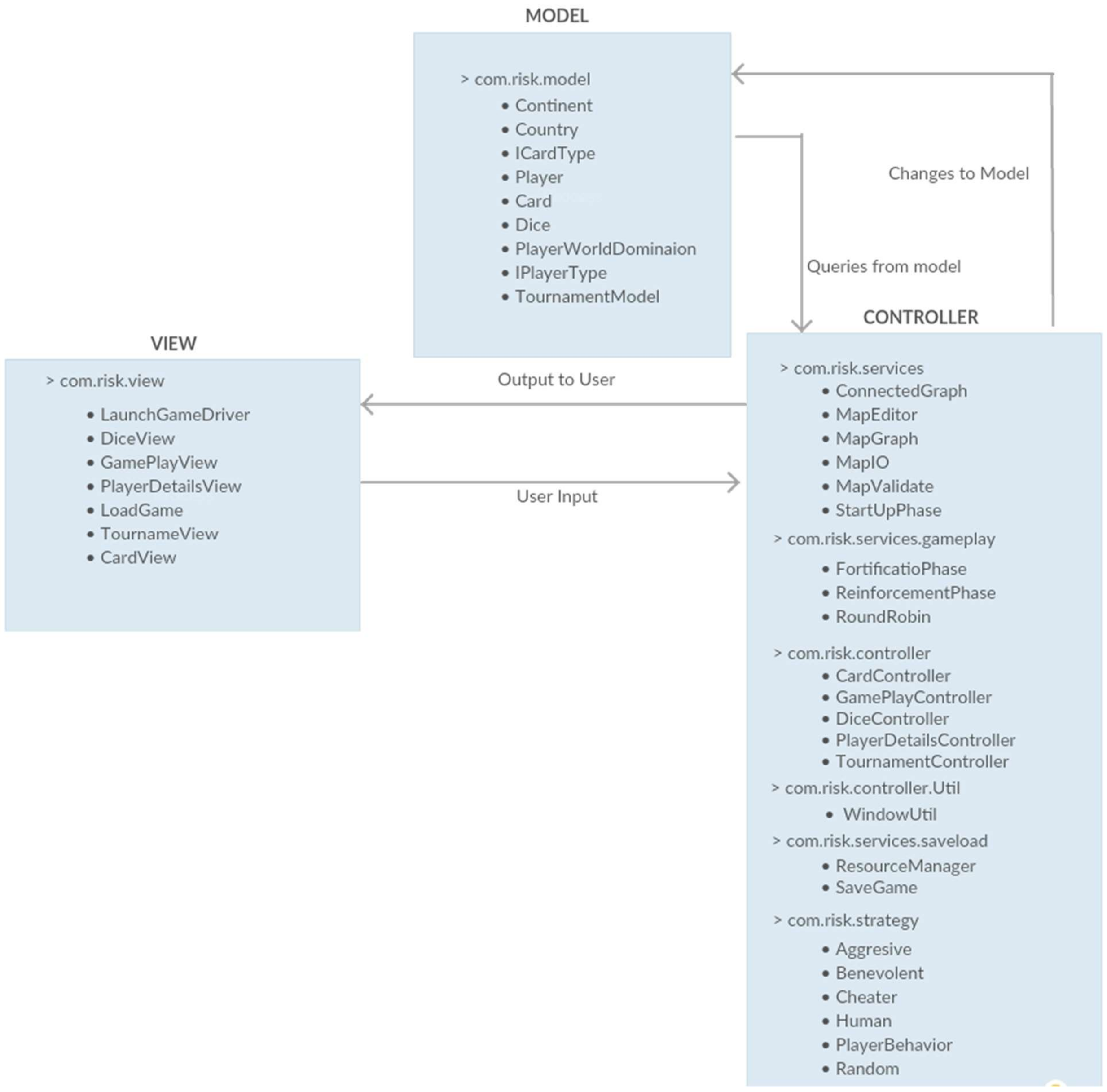
Architectural Design: Our RISK game is inspired Model View Controller (M.V.C) architecture, but it is not a STRICT implementation of MVC. We have divided and distributed our modules in the parts **MODEL, VIEW and SERVICES**. Below is the block diagram of our implementation of MVC architecture. In Model we have all the states of the game, like player, cards, dice etc. In our View, we have all the FXML loaders. No business logic is implemented in the view, neither view interacts with the models directly. Out business logic is under SERVICES folder, which can be related to controller in MVC but is not exactly the controller in MVC



Modules in Model:

We have 7 models described below and 2 interfaces:

1. Player: Player class is used to model the actual players of the game. We have implemented reinforcement, fortification and attack phase here.
2. Country: Country class model's country, 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, which can be taken as a superset of countries, where each country belongs to one and only one continent.
4. ICardType(Interface): It is used to model cards which are allocated after the attack phase if any player is eligible for the card.
5. IPlayerType(Interface): Interface to define type of aa player
6. Card: Card class is used to model cards, which are assigned to players after they win a country
7. Dice: Class to model dice objects for the attack phase.
8. PlayerWorldDomination: This class models the data to be displayed for the player word 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 for business login:

Our business login is implemented by services, controller and strategy folders. Implements the game flow, players strategies all the implemented by this module.

- MapEditor: This class provides functionality to edit or create a map from scratch.
- MapGraph: MapGraph creates a graph (connected) from map data and provides methods to modify the graph.
- ConnectedGraph: Checks whether a graph is connected or not.
- MapIO: This class reads the data from a **.map** file and provides it to MapEditor's object. It also gets data from MapEditor's object and writes it to an empty **.map** file.
- MapValidate: It is responsible for verifying the correctness of a map which is tobe loaded for game play.
- StartUpPhase: It takes data from MapIO, and initializes data for Players, countries and armies. Here countries are randomly assigned to each player.
- Gameplay: It has the following classes.
    - ReinforcementPhase: It has methods to get the number of armies calculated for are assignment to each player.
    - FortificationPhase: It provides method to pass the armies form one country to another.
    - RoundRobin: It provides functionality for round robin traversal among the players.
- Controller: It has controllers for controlling the models.
    - CardController: It governs the card view. It loads the card for the game play
    - DiceController: It governs the attack view. It manipulates the attack view and updates the dice objects according to the attack phase.

    - GamePlayController: It governs the game play.
    - PlayerDetailsController: It takes the number of players and, player names
    - TournamentController: Class to interact with the TournamentView FXML. It takes inputs from the user and calls Tournament model for operating the Tournament
- WindowUtil: It has various helper functions related for handling JavaFX features.
- Strategy
    - Aggressive: It governs aggressive players game phases
    - Benevolent: It governs benevolent players game phases
    - Cheater: It governs cheater players game phases
    - Random: It governs random players game phases
    - PlayerBehavior: Provides an abstract class for player strategies
- Saveload:
    - ResourceManager: It provides logic for loading and saving the game
    - SaveData: To test the logic of resource manager

Modules in View:

1. LaunchGameDriver: Provides an interface for the user to interact with the game. It launches the main window of the game to load or create the map file.
2. DiceView: This view provides a display window for the attack phase. Player interacts with this window during the attack phase.
3. GamePlayView: It is used to initialize the game play screen
4. PlayerDetailsWindow: View for the player selection. It provides options to select number of players and their names.
5. TournamentView: It basically has loads the tournament view FXML.
6. Card View: Load card view FXML before reinforcement.
7. LoadGame: Loads MapLayOutSelector.FXML for loading an already saved game.