Department of Engineering and Computer Science

Architectural Design

Advanced Programming Practices
Fall 2018

Team 3
Dmitry Kryukov
Ksenia Popova
Rodolfo Mateus Mota Miranda
Nikitha Papani

Concordia University
Montreal, Quebec, Canada

November 2018

In our project, we are using Model-View-Controller architecture. MVC - three-tier architectural model and it is widely used for many object-oriented designs with user interaction. The view consists of multiple Views that are implemented as an Observer pattern.

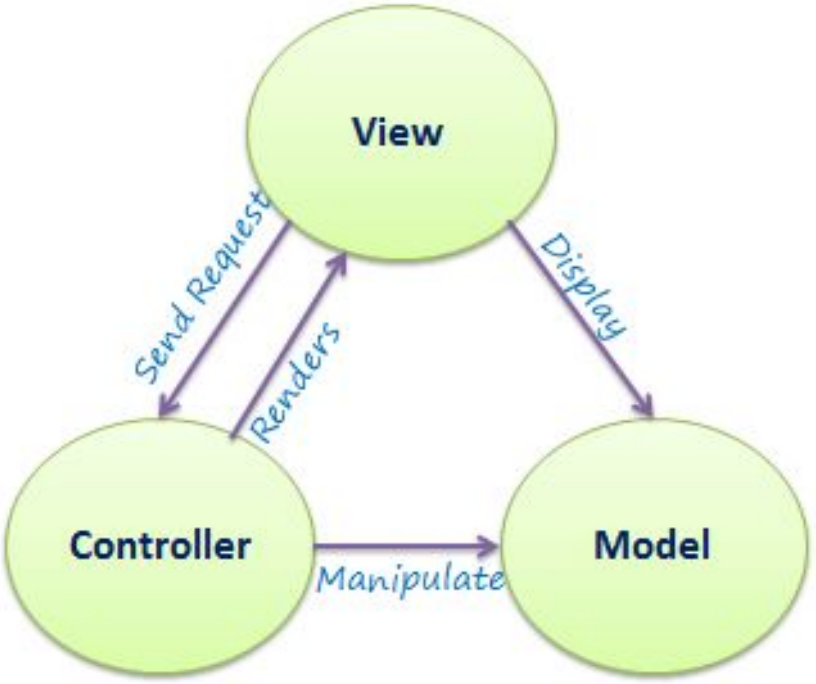


Image 1. MVC.

There are three layers of architecture in this architectural model:

Model - manages the behaviour of data (state) of the game. Model responds to the requests from the user about the state and to the requests to change the state. In our project we have multiple models responsible for the game data, such as map information (e.g. connections between countries, country's current owner), current game state (e.g. which player is the turn now, what phase of the turn is currently going), current amount of armies for each country, number of cards for each player. It provides the needed information to the View so it can be displayed to the user, and changes it if there is a request from the Controller to do so (e.g. change amount of armies). Model is represented by the classes defining the game entities.

View (game window) - there are multiple views in the project and all of them are implemented as an **Observer** pattern:

- Phase view: displays the name of the phase, current player, information about actions in the current phase. Observer.
- World domination view: displays the percentage of the map controlled by every player, continent controlled by every player (if the player controls any

continent), the number of armies of every player. Observer.

- Card exchange view: displays all the card owned by every player. During the reinforcement, phase player can exchange cards for armies according to the game rules. This view is active only during the reinforcement phase. The player can see the view during other phases, but can't exchange cards. Observer.
- Attack panel: contains the choice of the number of dices for the attack, button for the "All in" option.
- Dice panel: dices for the attack.
- Game view: displays the map, countries and connections between them. View requests the state information from the models and updates the information displayed on the interface if needed. User's actions with the game are sent to the Controller that makes calls to the Model. View requests the update from the Model and changes the information on the screen accordingly to the response.

Controller - Contains the main game logic, such as switch turn functionality, next phase, select and highlight countries, detecting the winner of the game, the end of the game, etc. accept input from the user and instructs Model how to act based on that input. Controllers translate user's interactions into actions to the Model that it can perform. Main game file game.java implements the **Observable interface.**

Package diagram for the game is represented below.

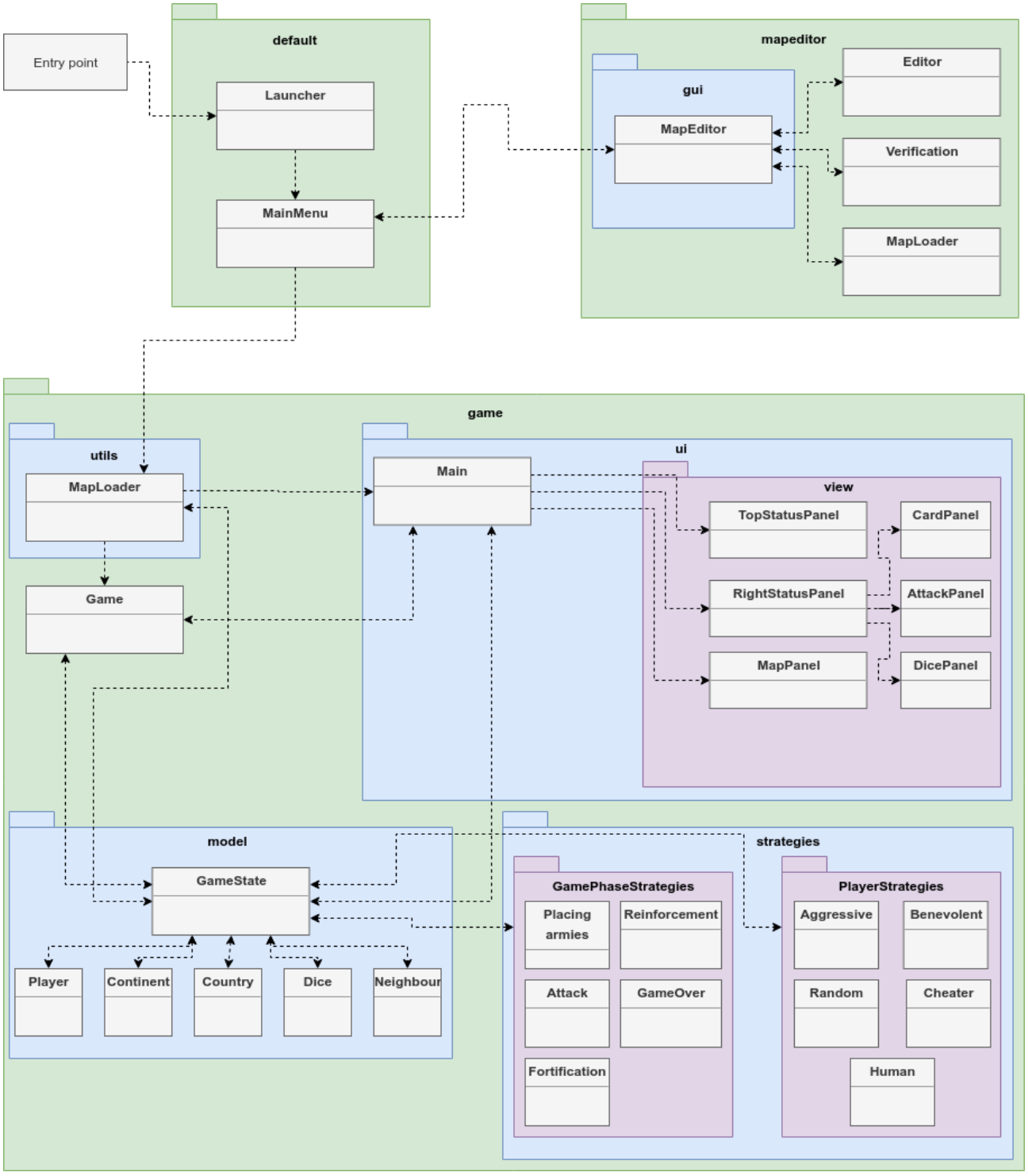


Image 2. Pakage diagram.

Default package contains the game launcher that can start a new game or open the map editor. The user can choose what he wants to do in the main menu of the launcher.

Map Editor allows to edit the existing map and to create a new map. When creating a new map, the user can specify all the necessary data for the map to be valid. When the

user tried to save the map after the map editing or creation, the map validation process starts. If the map doesn't meet some of the rules, the user gets notified and this map cannot be saved until he solves the problem.

The game package contains all the main files of the game, such as Game.java (Controller, implements **Observable interface**), model folder with model representations (Player, Country, etc.) and views (phase view, world domination view, main game view, attack panel view, dice panel, card exchange view, all implemented as an **Observer**).

When a new game is started, the main controlling functionality is contained in the Game class (implements the Observable interface). It launches the game flow. Generated views request data from the Model and Controller and display it. Controller (game) catches user's interactions with the Views through the game process and request data changes accordingly to these interactions. Views update respectfully to these changes showing updated data to the user.

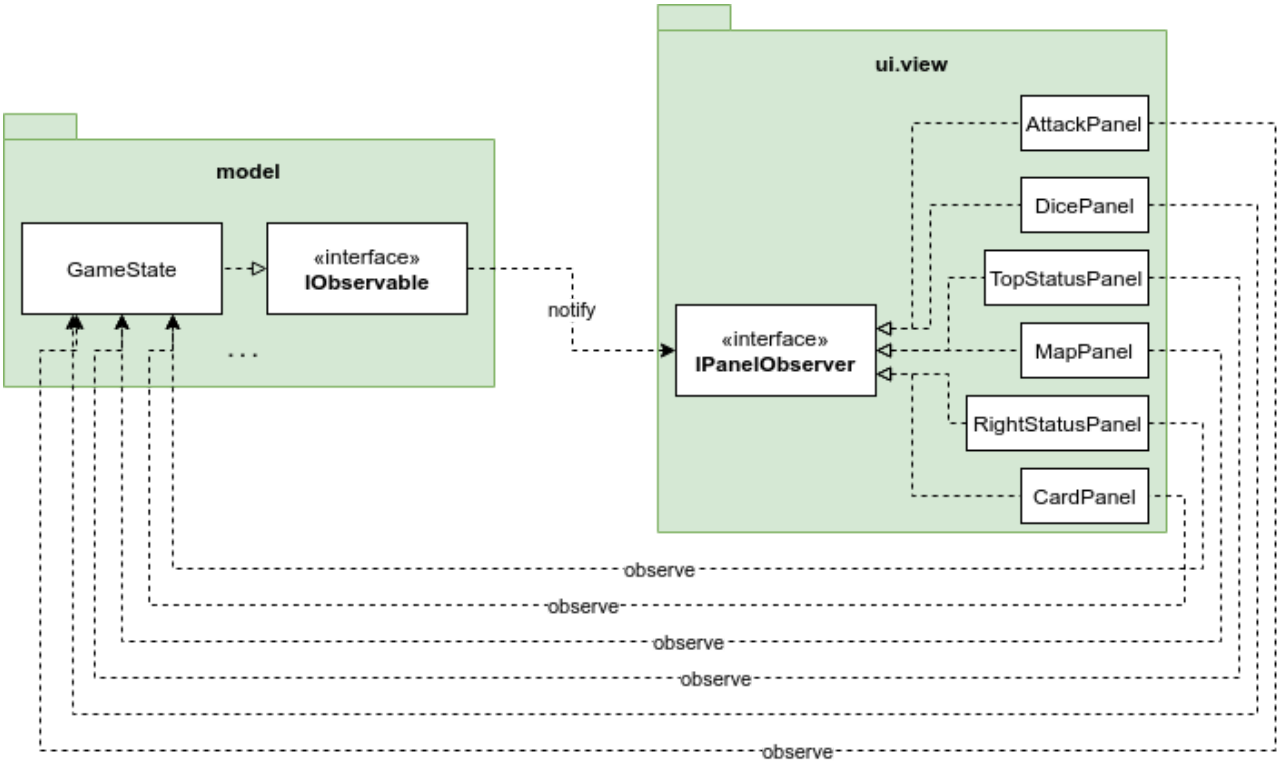


Image 3. Observer pattern.

Strategy pattern.

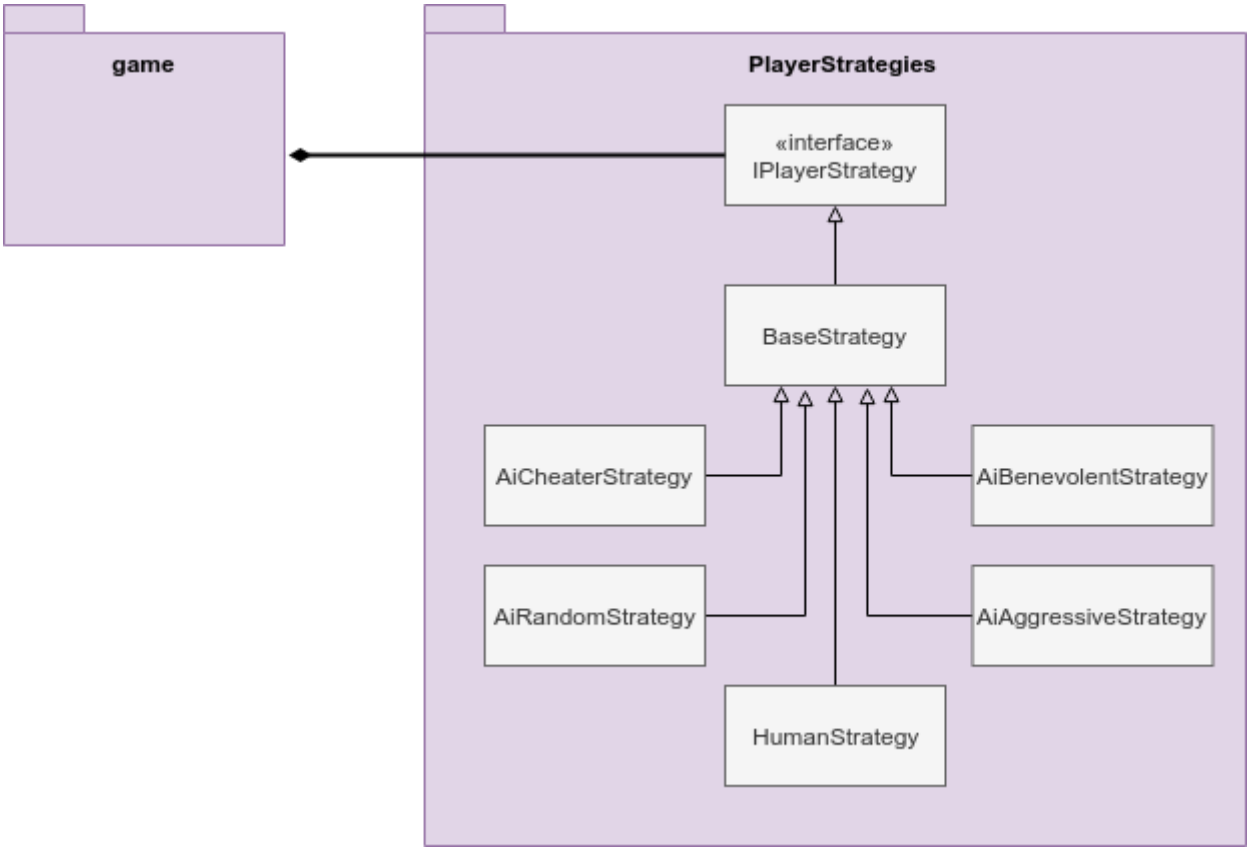
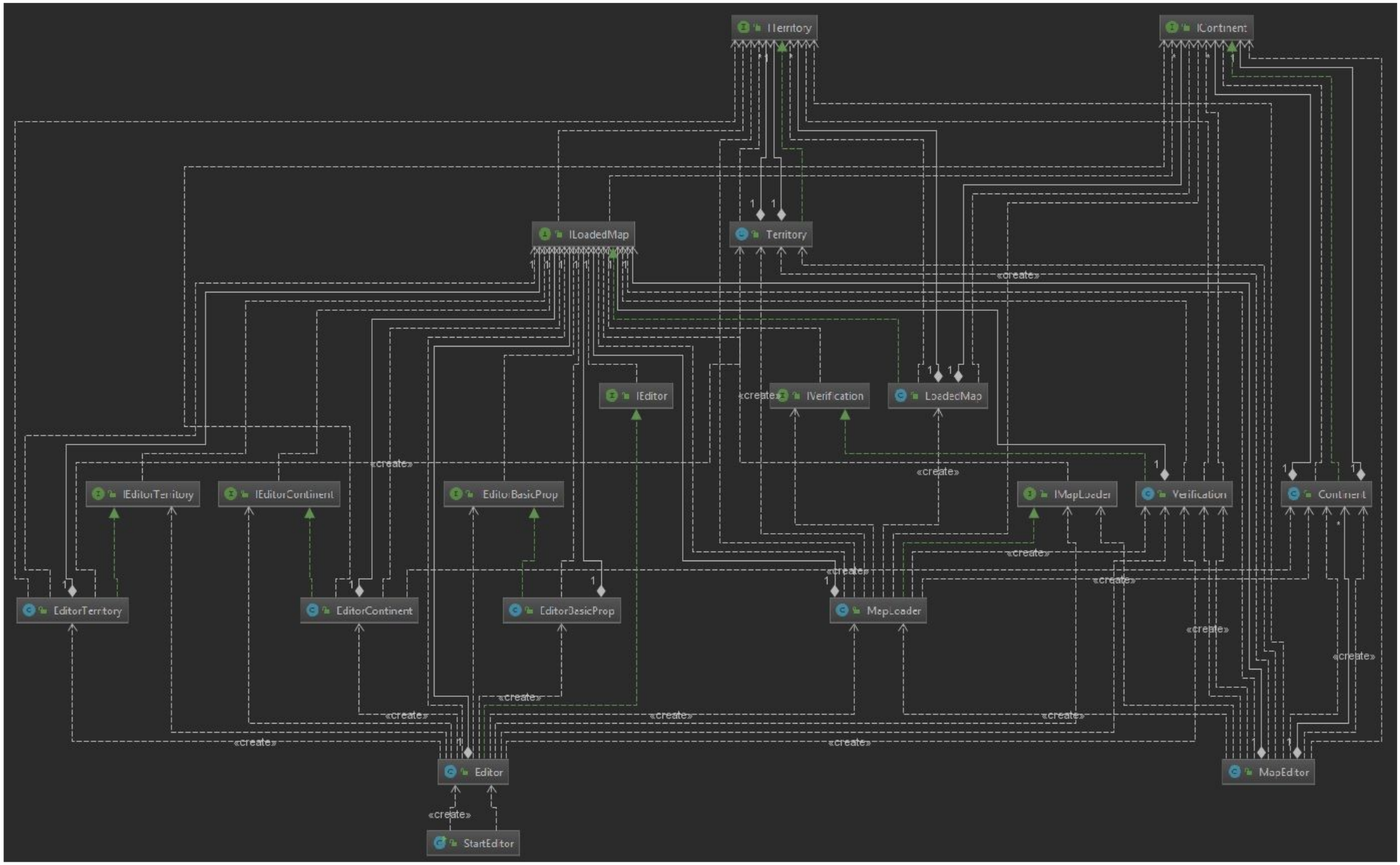
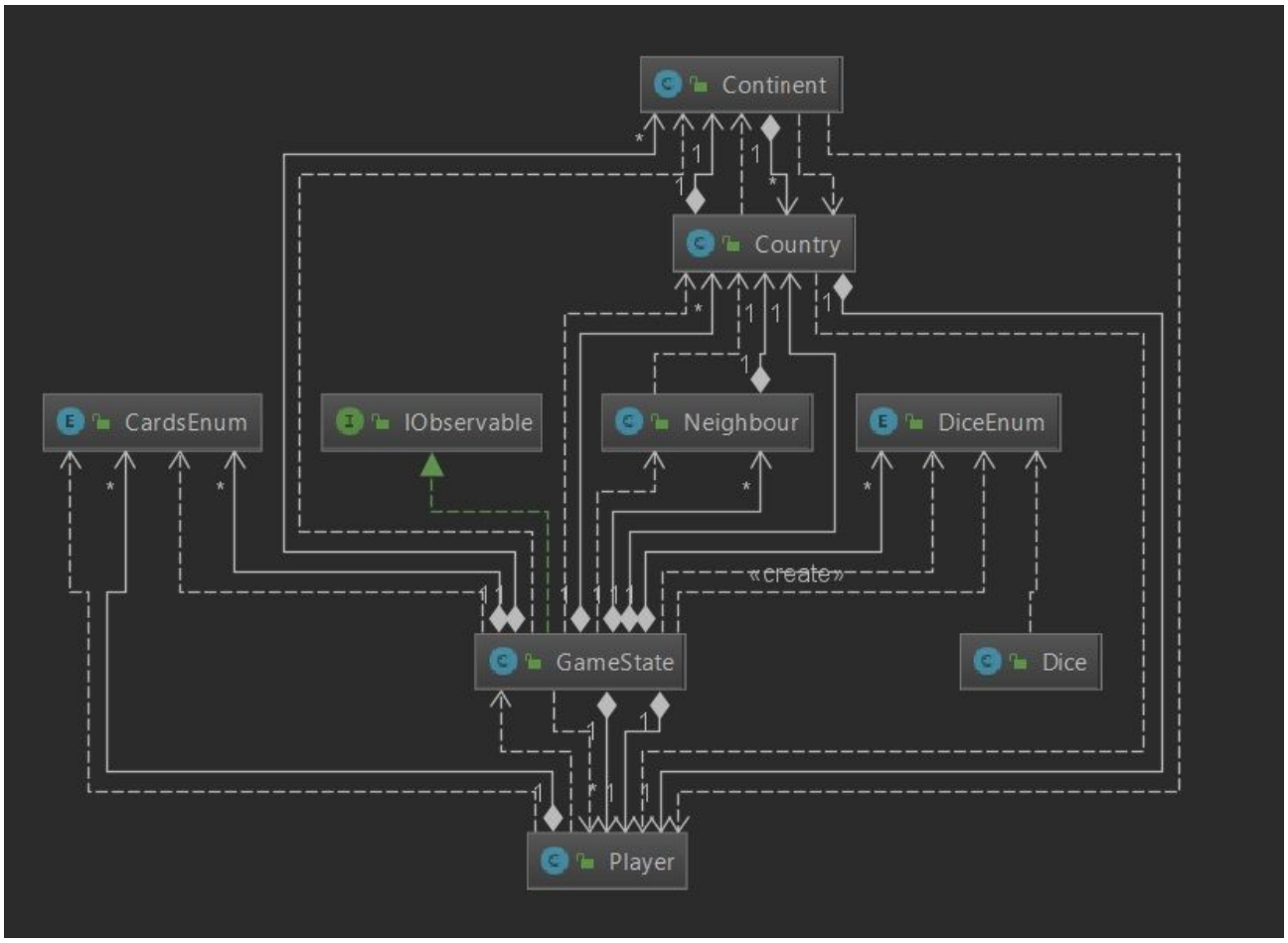


Image 4. Strategy pattern.

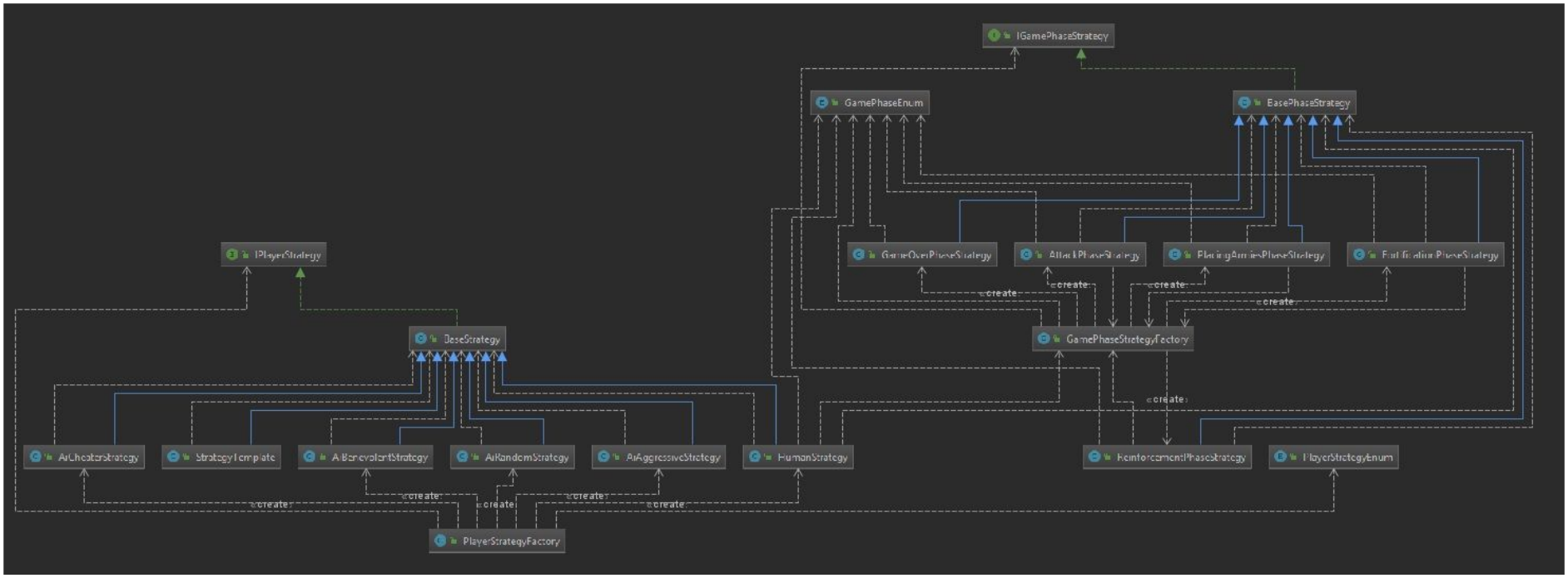
Built-in tools in the IDE allow you to create some charts that we put below.
Class diagram.


ITerritory
IContinent
ILoadedMap
Territory
«create»
IEditor
IVerification
LoadedMap
IEditorTerritory
IEditorContinent
IEditorBasicProp
IMapLoader
Verification
Continent
EditorTerritory
EditorContinent
EditorBasicProp
MapLoader
Editor
MapEditor
StartEditor

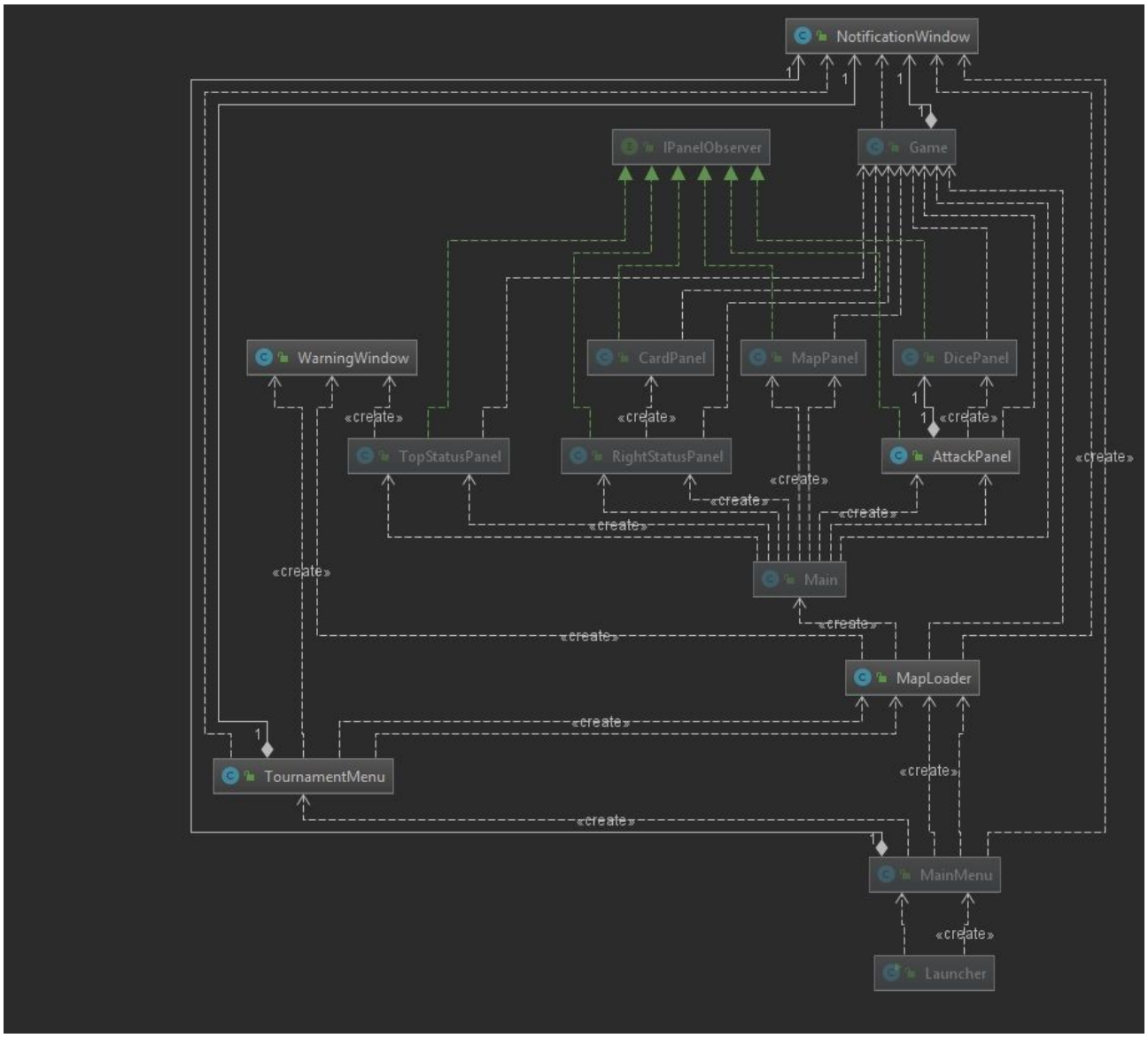
Models UML.



Strategy UML.


IGamePhaseStrategy
GamePhaseEnum
BasePhaseStrategy
IPlayerStrategy
GameOverPhaseStrategy
AttackPhaseStrategy
PlacingArmiesPhaseStrategy
FortifyingPhaseStrategy
«create»
BaseStrategy
GamePhaseStrategyFactory
AiCheaterStrategy
StrategyTemplate
AiBenevolentStrategy
AiRandomStrategy
AiAggressiveStrategy
HumanStrategy
ReinforcementPhaseStrategy
PlayerStrategyEnum
PlayerStrategyFactory

UI diagram.


NotificationWindow
IPanelObserver
Game
WarningWindow
CardPanel
MapPanel
DicePanel
TopStatusPanel
RightStatusPanel
AttackPanel
Main
MapLoader
TournamentMenu
MainMenu
Launcher
«create»
1