PROJECT DESCRIPTION

MVC implementation
Our implementation of MVC consists in Model + Controller on server side and View on client side. There is a VirtualView on the server side that virtualizes the view, which keeps the state of the game, and its attributes are ready to be sent to the clients.
Clients can't verify the information inserted by the user, the server elaborates them for security reasons.

Server
In our Model we use Manager Classes to manage information about each aspect of the game. Each Manager contains a private Class (for example: Island in a IslandManager) for information hiding: it keeps the information about his private class and communicates with RoundStrategy or Game.

When an information is changed in a class, the change is notified to the VirtualView class in Controller, which implements listener interfaces.

To create a game in expert mode, we have used a factory pattern which creates three random specials. After the creation of the special game, when a special card is played, we use Strategy pattern to override the normal Round.

In case of an equal number of towers and professors, the player's team who is positioned in the lower index cell in the 'players' list in PlayerManager wins (we assume this because it is not specified in rules).

A player can't chose his team because the game choose it for him:

- In the 2-player game, the first player is on the white team, the second player is on the black team;

- In the 3-player game, the first player is on the white team, the second player is on the black team and the third player is on the gray team;

- In the 4-player game, the first and second player are on the white team, the third and fourth are on the black team;

Controller is interfaced with the Model by an interface that allows it to only use methods that can change some attributes on it and the setters for the listeners.

The first client to connect to the server receives an answer that allows it to set up a new game (number of players and game mode).

If there is a game saved on file, the first client to connect receives the game setting of the previous game, and he can choose to restore it to create a new game.

If he chooses to restore it, the server expects a message with his previous nickname; if the user doesn’t remember his previous nickname, there is no way to go back and create a new game: he has to disconnect and try again.

Our implementation doesn’t explain the rules of game to the user, we assume he just knows them (when the user makes a move which is not allowed, the server notifies that to the client, that shows the error ‘move not allowed’).

We have used a proxy server side, to communicate with the clients. Client connections are stored in a proxy like VirtualClient.

Persistence
VirtualView is like a picture of that moment of the game: it saves every change and attribute in the relative inner class (for information hiding). After the planning phase and after the action phase of every player, everything is written on the saveGame.bin file.

Client
Client starts the connection asking for an ip address and port number. Then it asks if the player wants to use CLI or GUI.

Client uses a proxy to communicate with the server. It sends messages to the server and it manages all the answers received.

View
View contains all the game information and notifies the update to CLI or GUI using listeners. Informations are set with answers from the server.

CLI
CLI is the Command Line Interface, it prints on screen all the game’s information and all the updates from player’s plays using Listeners.

GUI
For the GUI, we used JavaFX. We have a main GUI class which extends javaFX Application class. It is launched by the Client class, if the user chooses to use it.
The GUI class implements the listeners interfaces, and receives all the notifies when there’s a change on the View. It then accesses the correct SceneController to modify the scene.

There are various SceneController classes with the respective Scene, one for each phase of the game.

SceneControllers can call directly the Proxy of the client to send information to the server, following a user interaction.

Students are moved by clicking on the student to move, and then on the place (table or island) where we want to move it. Mother nature is moved by clicking on the island where we want to move it.

Clouds and Assistant cards windows can be displayed in every moment of the game, by pressing a button.

When playing in expert mode, a “use special card” button is displayed. The scene can only be opened if the player hasn’t played any special card yet in his turn.
In the Special scene, after choosing a special card, the choice needs to be confirmed by pressing a button.
When choosing special cards that cause student exchanges, students need to be chosen one by one, and then confirmed using the “Confirm special” button on the MainScene.
For special 7, it is done by adding them using the add button and then selecting the students to exchange from the entrance.
For special 10, by choosing them from the table and the entrance.