PROJECT SANTORINI

HOW TO USE:

In terminal besides "java -jar *path-to-jar*" the execution needs a specific parameter:

- "server" to run the server
- "CLIClient" to run a client with command line interface
- "GUIClient" to run a client with graphic user interface

For example to run the server "java -jar *path-to-jar* server".

DESIGN CHOICES:

The project is based on MVC pattern in which in our case M and C are server side while V is client side (thin client architecture). Now we will analyze our design choices for both side.

SERVER SIDE:

We considered M of Model as a finite-state-machine, we tried to implement it in the most literal way. To do that we used several patterns:

1. STATE PATTERN – this allowed us to create the states of the FSM without the transition logic, in our case the server is never waiting and every state might evolve in response to an input.
2. VISITOR PATTERN – this allowed us to completely remove the parsing from the Model. It allowed us to define operations without changing the classes of the element on which it operates.
3. OBSERVER PATTERN – this allowed us to notify the changes of Model to the View avoiding any parsing on model side, every parsing is made by controller. The observed are the states while the observer is the controller. We chose to not use an interface for the observer because it's unique.

To complete the implementation of the FSM we created an XML table that implements the transition logic between the states and practically it defines the game flow. This allowed us to change dynamically the flow of the game. By changing this XML we can change the flow without modifying any line of code. This implementation guarantees the possibility to create the most various different game rules possible.

To implement the power of gods we created an XML file that contains name and description of the power. This permits us to associate power with card, to add unique gods with multiple powers and it gives us possibility to create cards with new power by combining already existing ones only by modifying XML file.

We used also DECORATOR PATTERN to implement the special rules (part of god powers such winCondition or special movement or building). To manage the remaining part of god powers we used Booleans that permits us to combine power in a possible future.

The C of Controller is used to parse or deparse every message between Model and View. Every player has its own controller.

The implementation of communication is made by the Handler class and the association between Handler and Controller is made by HandlerHub class which is an hub that could decide to send the message in single communication, broadcast or semi-broadcast. After the connection with the server the non-playing players are in the globalHub (an HandlerHub instance), this permits us to manage multiple games together with LobbyManager .

CLIENT SIDE:

In the V we used the COMMAND PATTERN to encapsulate the request as an object, this allowed us to add or remove a command very easily. A command is generated from an action of the user or from a server message. We have a parsing and deparsing unite separated by interfaces and multiple command managers that manage command from a specific interface. This system works for BOTH CLI and GUI.

We recommend to set the 3D camera from the upper angle, but you can use every possible angle.

We didn't put any application logic in the client but only communication logic.

CONCLUSIONS:

The project is easily extendable, modular and it can be changed very easily. We tried to make the most possible lean structure for classes and methods preferring multiple classes but smaller to big classes (with too different methods).