Strategic Game: CElization

Name: Pardis Pashakhanloo

Student ID: 91106449

Brief note about the design pattern

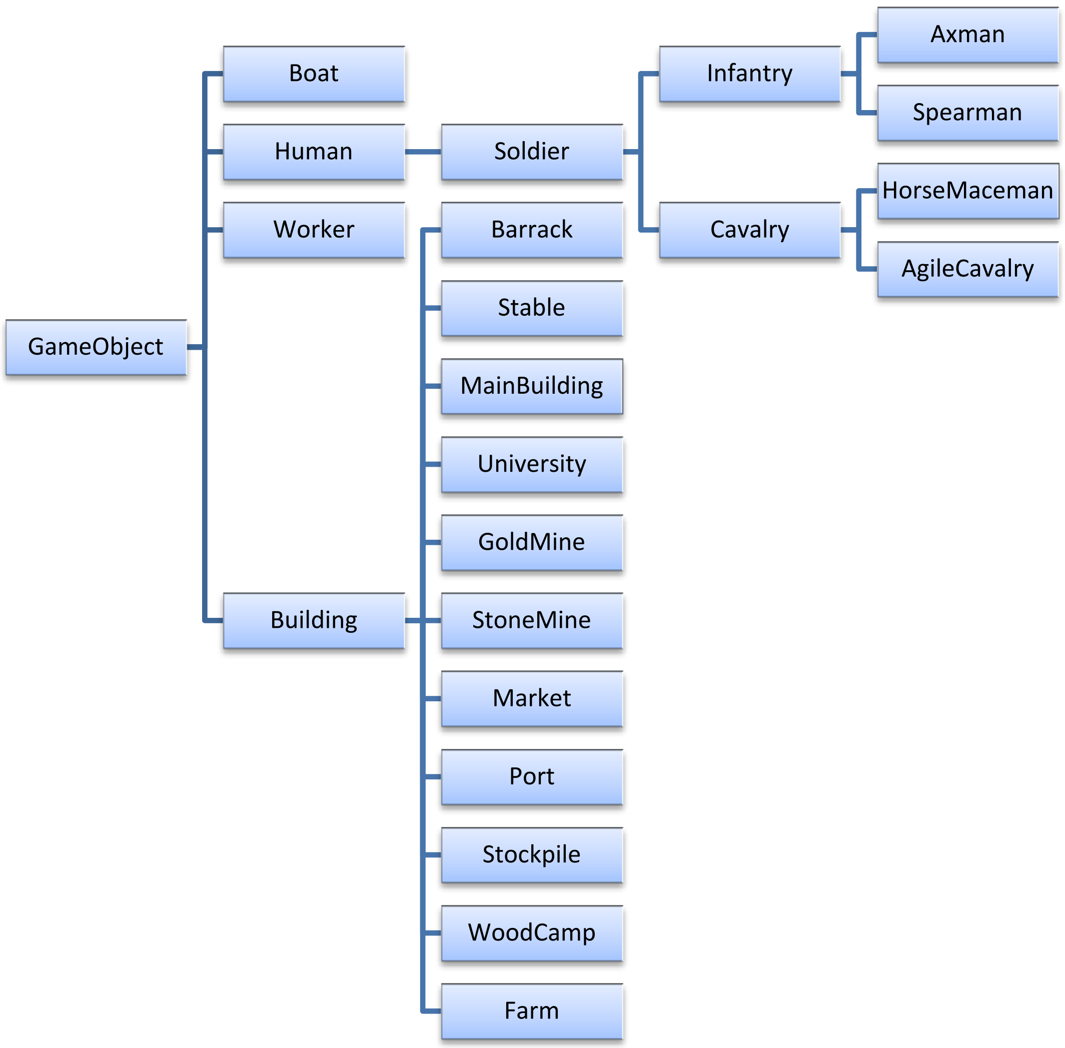
First phase

Main Classes

GameObjects

In the first phase, I imagined each unit of the game as a GameObject and designed a hierarchy for them.

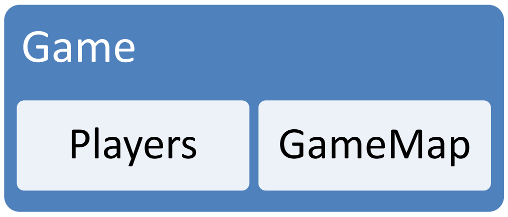
As you can see below:



While working on implementing actions, I added a tagging interface Movable to my code as I had to group Boat with Human and it was the method I used.

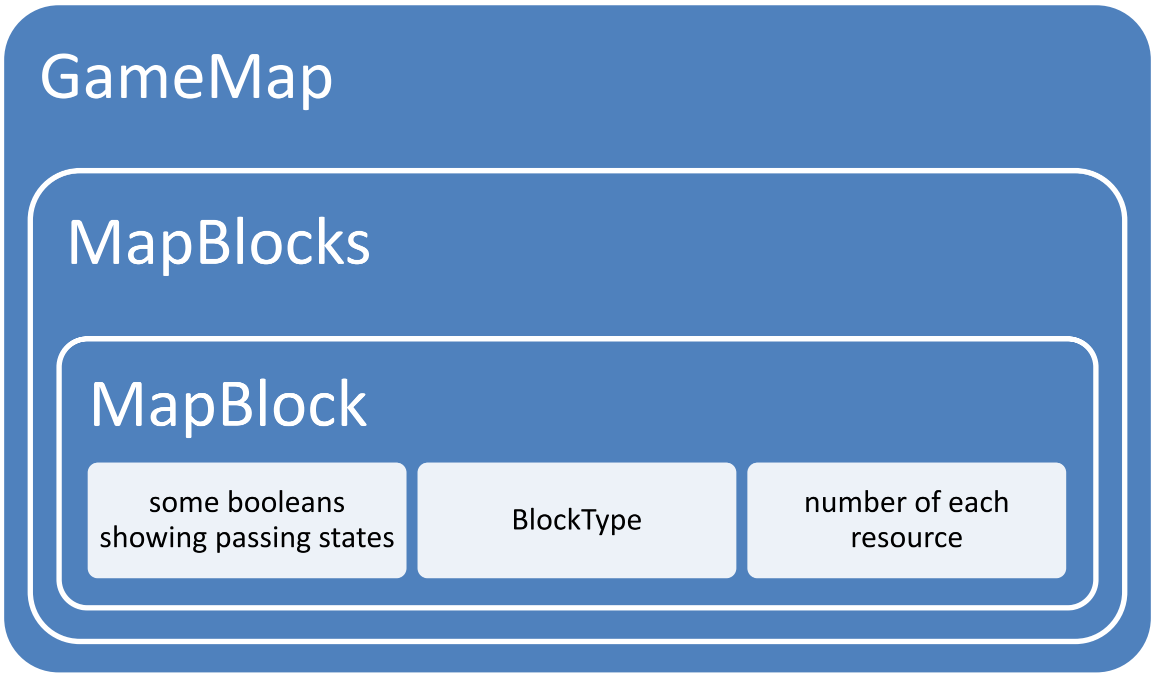
Game, GameMap, and MapBlock

Then, I designed a main class Game which keeps Players, GameMap, and also assigns a primary id to each Player and GameObject. I made it singleton as I had to have only one Game for the whole game. The method for saving and loading is also placed in this Class. The reason will be explained further.



As the game needed a map, I designed GameMap. GameMap consists of a 2d array of MapBlocks. GameMap can either generate a random map or generate a map out of five 2d arrays, indicating the type of block and the number of each resource in each block. While a random map is generated, the type of the block is considered.

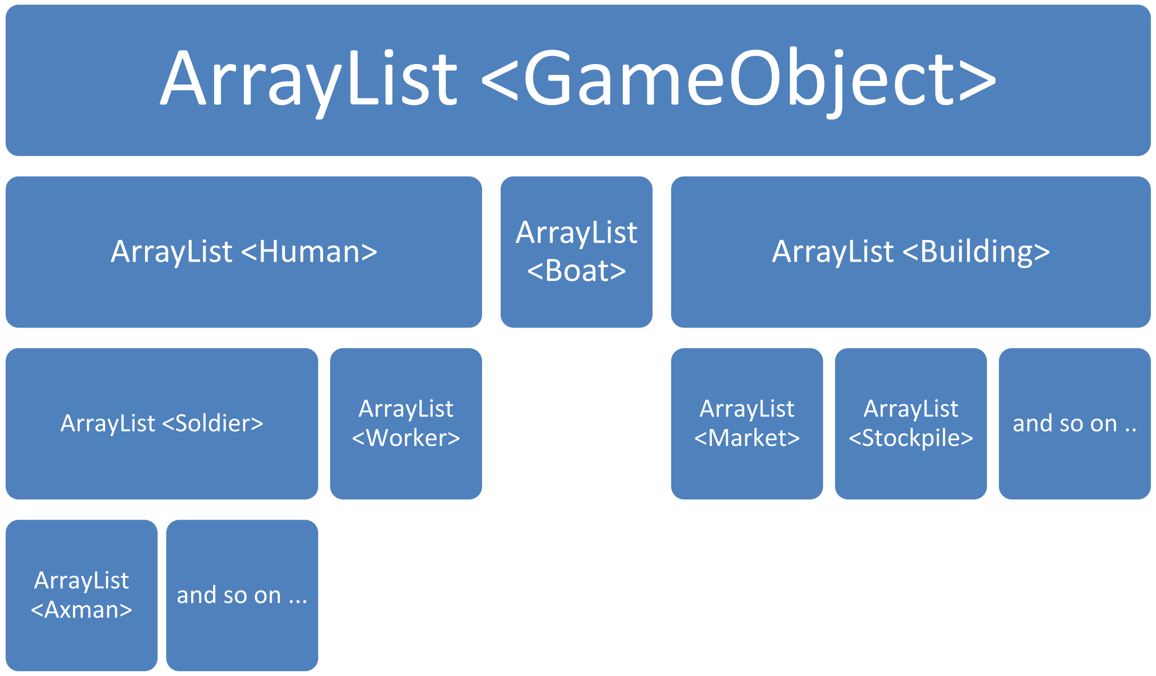
Next class is MapBlock. Each MapBlock keeps the number of each resource it has, its type, and some Boolean variables showing whether Movable GameObjects can pass it or not.



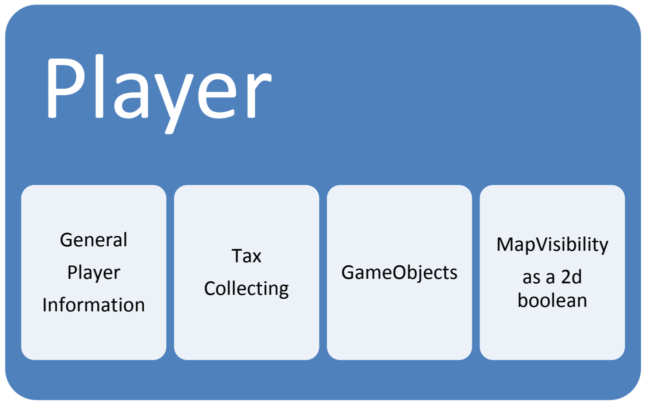
Player

Player is the next essential class of my project. Player keeps all of its GameObjects in a categorized way, as you can see below.

When a new GameObject is instantiated, it is automatically added to these array lists.



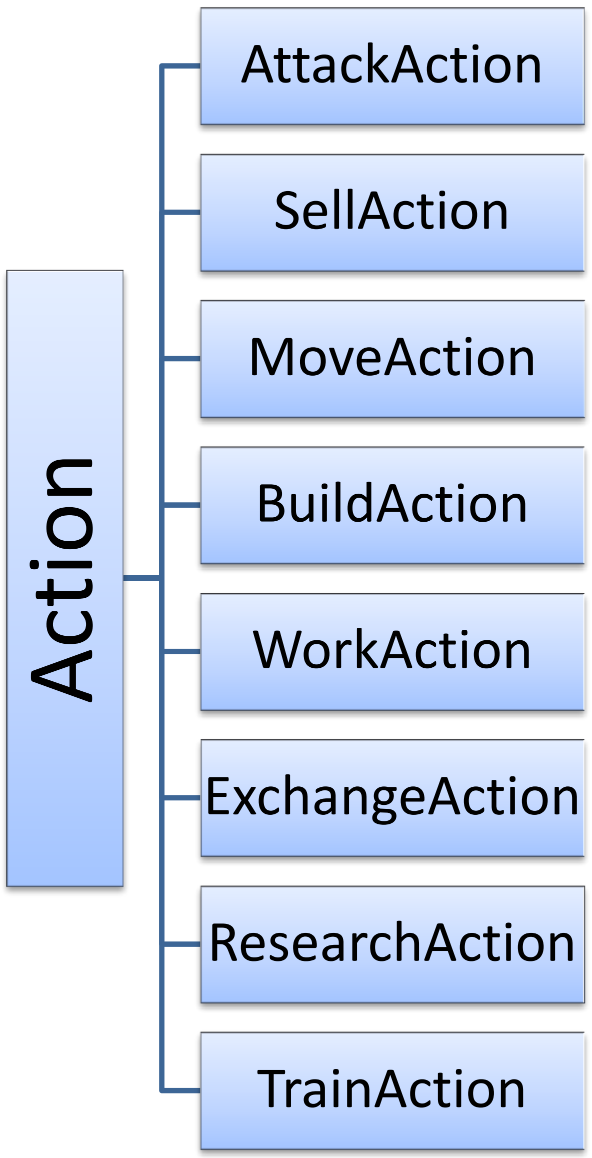
In this way, each category and each sub category is accessed easily and much of useless code for doing operations on them is removed. MapVisibility – or map fog in some games – is also kept in Player.



GeneralPlayerInfo as it is obvious from the name, has the duty to keep information related to each player. I had to utilize such class because there was a huge amount of information which had to be kept. It also initializes the amount of resources, turns to build, researches that each player must do and so on. It keeps the status of the ability to build or train GameObjects as well as production rates. In other words, it works as a database of all information which are constant or variable for one Player.

ActionControllers and Actions

While thinking of a method to differentiate between a one-turn action and a couple-of-turns action, I came up with the idea of making ActionControllers and Actions. Each Action, does a one-turn action. But each ActionController controls the action each turn must be done on a GameObject. These two concepts have a hierarchy as shown below:



Let’s talk more about some of the Actions.

Each Action has a main method, called doAction(), which is the only method invoked by its Controller.

BuildAction and TrainAction

These two Actions has the only duty of making a new GameObject (e.g. new Building() or new Human()).

WorkAction and AttackAction

These Actions is empty, because there is not any one-turn action related to them.

MoveAction

This Action operates BFS algorithm on the map and move a Movable GameObject on block nearer to the destination and makes the map visible around that object.

ExchangeAction and SellAction

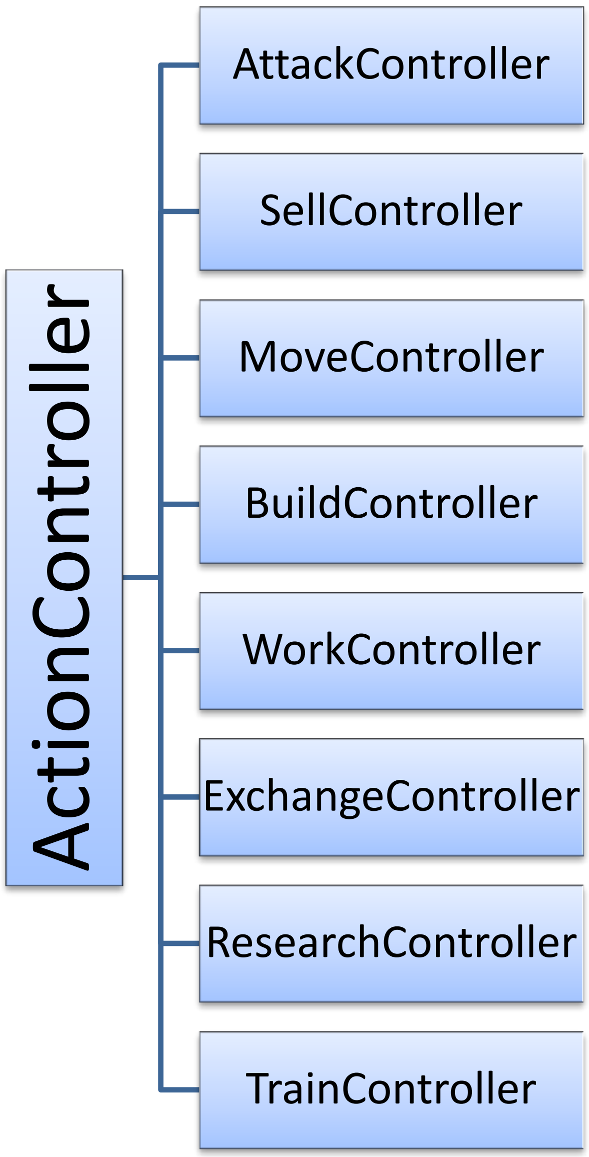
These Actions just have to do with calculating the number of resources which are added to or removed from the player's stock.

ResearchAction

This Action enforces the effects that a research has on the player's status.

ActionControllers, control action flow of as well as checking the justifiability of them. Each Controller has a nextTurn() method.

The hierarchy of ActionControllers can be seen below:



Brief explanation about ActionControllers:

AttackController

NextTurn() of this Controller, checks whether an object is neighbor of the player's soldier or not. If not, it can operate the attack and calculate the health and the score of both sides.

ExchangeController and SellController

NextTurn() of these, only operate doAction() of their Actions.

MoveController

While an object is in the process of moving to a destination, the ability of passing the blocks and checking if it has reached the destination is the duty of nextTurn() of the MoveController.

BuildController

This Controller checks build flow. A Build has three steps: reaching around the building which is about to be built, decreasing resources, and building the building after intended turns. The first step hires a MoveAction each turn until reaching the build place.

ResearchController

This controller makes a research status as finished, change available researches and decrease resources.

WorkController

This Controller checks work flow. A Work has three steps: moving to workplace, working in the Building, moving to a main building or stockpile to unload the cargo, and increase resources.

TrainController

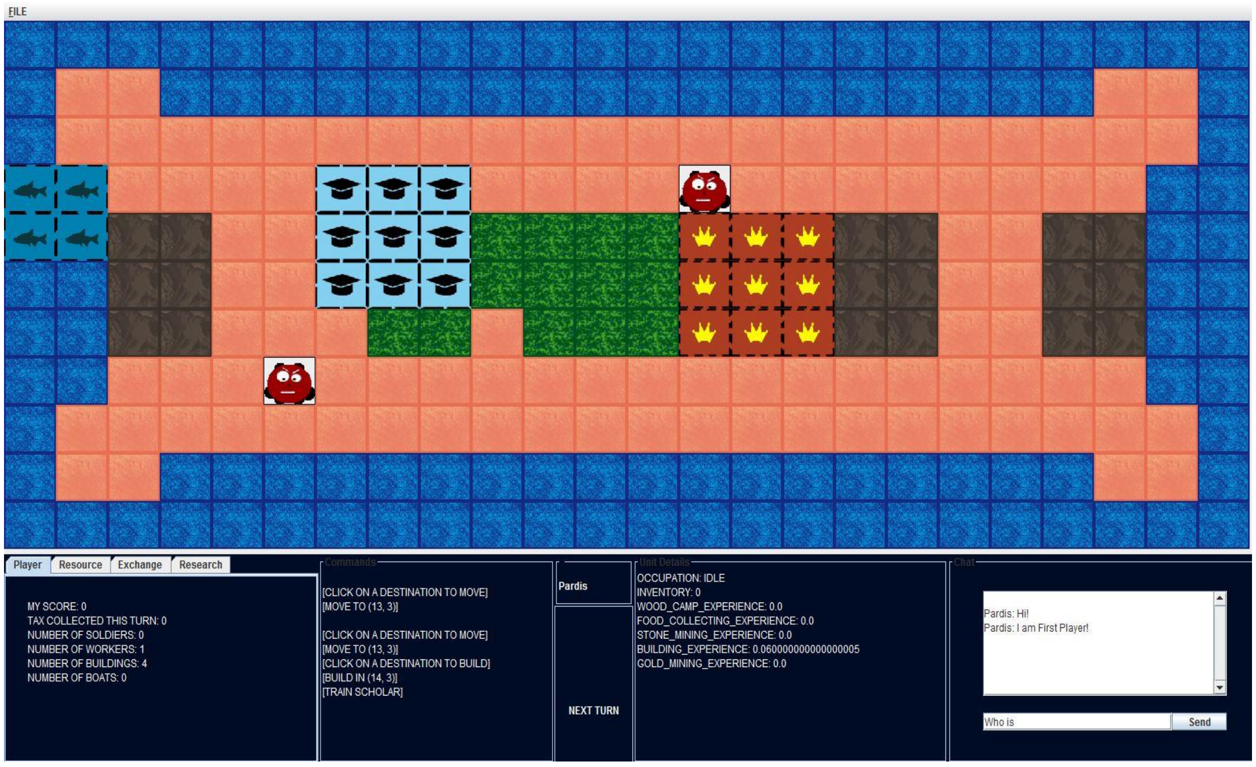
This Controller invokes doAction() after remaining turns equals zero and decrease resources.

Other classes

Resource, Research and Point are other classes which are sort of boxes to save data.

Second phase

Added GUI



```
MY SCORE: 0
TAX COLLECTED THIS TURN: 0
NUMBER OF SOLDIERS: 0
NUMBER OF WORKERS: 1
NUMBER OF BUILDINGS: 4
NUMBER OF BOATS: 0
```

```
[CLICK ON A DESTINATION TO MOVE]
[MOVE TO (13, 3)]

[CLICK ON A DESTINATION TO MOVE]
[MOVE TO (13, 3)]
[CLICK ON A DESTINATION TO BUILD]
[BUILD IN (14, 3)]
[TRAIN SCHOLAR]
```

```
OCCUPATION: IDLE
INVENTORY: 0
WOOD_CAMP_EXPERIENCE: 0.0
FOOD_COLLECTING_EXPERIENCE: 0.0
STONE_MINING_EXPERIENCE: 0.0
BUILDING_EXPERIENCE: 0.060000000000000005
GOLD_MINING_EXPERIENCE: 0.0
```

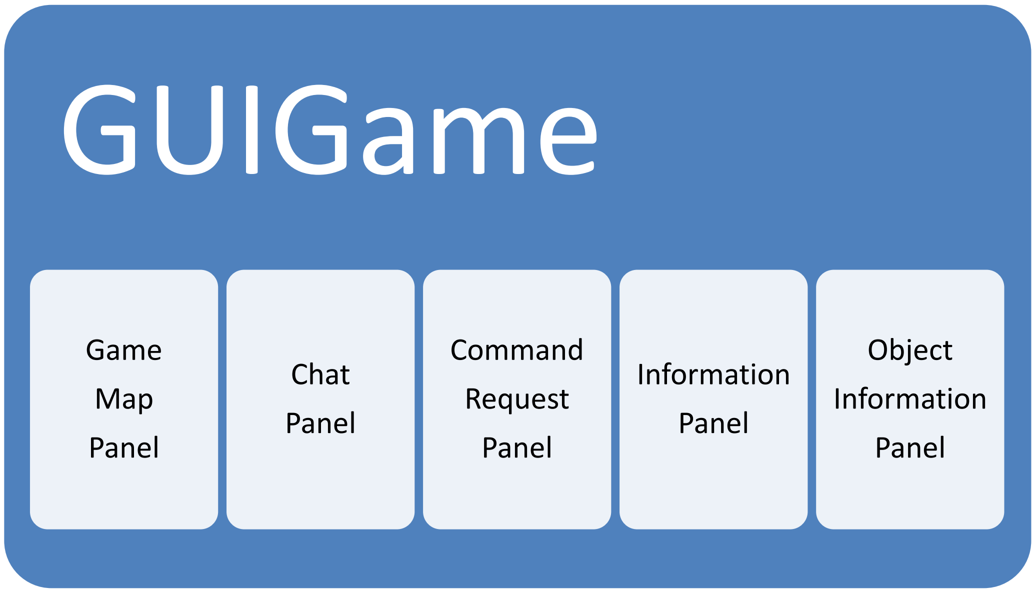
```
Pardis: Hi!
Pardis: I am First Player!
```

In the second phase, I wanted to use MVC design pattern in order to relate logic part to GUI part. But after calculating the time and encountering couple of problems, I changed the pattern.

Main Classes

GUIGame - JFrame

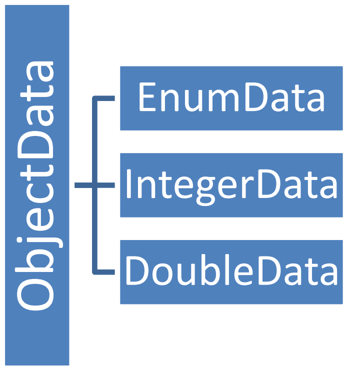
This class extends JFrame and is the main frame of the game. It has five main panels and a menu bar.



This class analyses request depending on their CommandTypes and show an informing message on CommandRequestPanel. Reconstructing the GUI is also the duty of a method in this class.

GUIGameObject

GameObject which were used in logic part, is different from GUIGameObject. Each of them has only a Type, ID, and a `java.util.`Map of object information. Data is kept related to its corresponding type, as shown below:

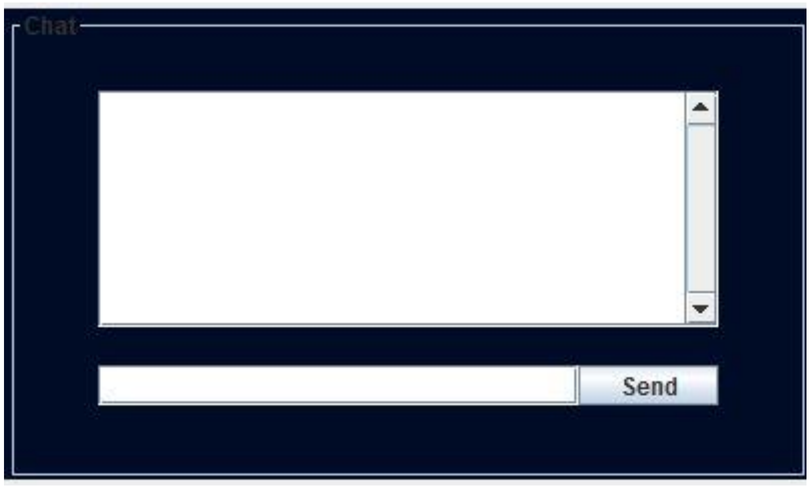


Other panels and classes

It is consists of GUIMapBlocks which extend JLabel. Each block has a ActionListener which opens a popup menu after right clicking. Then in the popup menu, an object on that block can be selected. There are choices of actions for each object which are shown as menu items in the popup menu. There is also another ActionListener that shows the details about a selected object in ObjectInfoPanel. The opening of a popup menu is controlled via a MouseAdapter.

Chat Panel

Each turn, the player which is its turn, can send a message so that other players can see it.



Type - enum

Type is an enum, which keeps an image, some CommandTypes, and some InfoTypes for each type of object. A type of resource or a block type os alco regarded as a Type.

Save Game

Saving a game can be done by choosing Save in menu bar. It asks for a directory and saves the whole game, via OutputStream.

```
FileOutputStream saveFile = new FileOutputStream(file, false);

ObjectOutputStream save = new ObjectOutputStream(saveFile);

save.writeObject(gameSave);

save.close();
```

Load Game

```
FileInputStream loadFile = new FileInputStream(file)   ;
ObjectInputStream loadObject = new ObjectInputStream(loadFile);

Object loadedGame = loadObject.readObject();

game = (Game) loadedGame;

loadObject.close();
```

It can alse be done by menu bar. It uses InputStream to load a game.

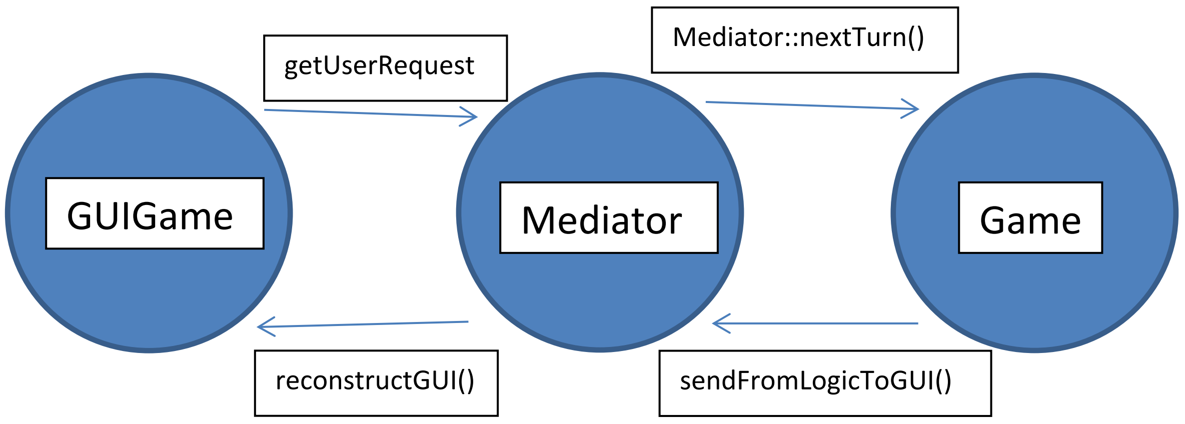
Save and Load are put in the Game class. Because, Game is singleton, so the getGame() of it is static. Static fields cannot be saved using a OutputStream. So a copy of it must be newed and saved. It can be done only inside the Game class, as its constructor is private.

Request Flow

A user can request a command via panels (ExchangePanel for exchanging, ResearchPanel for doing a research) or by right clicking on objects.

After the user presses the button on a panel or right clicks on an object and chooses a command, it is sent to the method of analyzeRequest() which is located in GUIGame. In analyzeRequest() a GUIObject and a CommandType is taken and depending on the CommandType, a UserRequest is made. Some of them need two clicks and two popup menus, like work, build, and move which need their destination to be distinguished. After completing a UserRequest, it is sent to UserRequests ArrayList which is located in GUIGame. An informing message is also shown on CommandRequestPanel. After NEXT_TURN button is pressed in the game, nextTurn() of GUIGame is called. It clears the ArrayList of requests and call the nextTurn() of the Mediator. Mediator has the duty to link logic to GUI and vice versa. It iterates over objects and invoke getUserRequest() of the Mediator for each request which has come from the object. This method makes an ActionController for each of the UserRequests. For example it makes a MoveController for a MoveRequest. It then link each GUIObject to the corresponding GameObejct using their IDs. It finally assigns the made ActionController to the corresponding GameObject. In the end, the connection between logic and GUI is completed by invoking the nextTurn() of the Player. It then checks some events that must happen each turn, like adding food by boats, collecting tax, and calling the nextTurn() of ActionControllers which are not null. Then the nextTurn() of the Game is invoked. This method assigns turn to the next player by startTurn() of the player. startTurn() somehow traverses the mentioned flow inverse. It calls the sendFromLogicToGUI() method of the mediator. It constructs GUI again, for this player and shows changes that has happened in one turn for this player, shows done actions in this turn in ObjectInfoPanel, shows the current amount of resources in ResourcesPanel, and reconstructs GUIObjects from their corresponding GameObject or block and assign the same ID to them.

The relationship among GUIGame, Mediator, and Game which are all singleton:


getUserRequest
Mediator::nextTurn()
GUIGame
Mediator
Game
reconstructGUI()
sendFromLogicToGUI()