**CONTROLLER behavior**

The Controller is made out of 3 different managers:

* The FaithTrack Manager, which “acts” on the FaithTrack, the players positions.
* The **Action Manager**, which is the “main” part of the controller. It receives events/messages from the client side, and performs a determined action on the model.
* The GameManager, which “acts” on the Turn, the CurrentPlayer, the Idle players, the Status of the game; it can give services to the other two managers in the controller.

For example, the ActionManager may need a player to advance. This is done by calling a faithTrackManager.advancePlayer( Player ).

**ACTION MANAGER:**

* The action Manager contains a set of **possible actions**:
  1. Choosing LeaderCards [starting phase]
  2. Choosing a Resource [starting phase]
  3. Activate a LeaderCard
  4. Discard a LeaderCard
  5. Activate the Production
  6. Change depot configuration
  7. Asking the game “which cards can I buy?”
  8. Buy a card and placing it in the slot (8)
  9. Grabbing resources from the market
  10. Choosing what do to with a specific resource that was grabbed in the market (9)
  11. End Turn
* There are also some helper methods. (Lorenzo move, for example)

Some actions may require two or more steps (7) (9). Some actions might fail. Some actions may the player to choose something specific (1) (2) (8) (10).

The controller “communicates” with the client through the model, enabling certain objects (called Middle Objects) that must be solved by the client.

* There are **7 Middle Objects by the Server Side**:
  1. Vendor of the Development Cards (contains the cards, and the slots)
  2. Error Object (used to send error messages to the client)
  3. Leaderboard (used to communicate that the game has ended)
  4. LeaderCards Object (used to ask the client to pick 2 cards) (contains the cards)
  5. Resource Object (same as (4), but with a resource)
  6. MessageHelper (used to send notification messages)
  7. MarketHelper (enables a complex choosing strategy for action number (9)
* There are only **5 Middle Objects in the Client Side**:
  1. The first missing is the Error object: the error message is showed to the client on receival and that’s it.
  2. The second missing is the messageHelper: for the same above reason.

When the ActionManager is invoked, it is because the Server has received a MESSAGE from the client.

The Message type would be either MSG\_INIT\_xxxx or MSG\_ACTION\_xxxx.

The Message is passed to the onMessage( MSG ) method, which is basically a filter for messages.

It first checks that the player who’s requesting something is the currentPlayer, and if so, calls the correct method of the 11 described above.

Every Action follows a standard pattern, defined as:

* Invoked with: Action\_xxx(Player player, MSG\_xxx)
* Message Validation : the message must be well formatted.
* Model Validation : before modifying anything on the model, it must be checked if the player can actually do the action.
* Model Manipulation : if the 2 above are passed, the model gets modified. Middle Objects may be enabled, depends by the action. Every change is notified to the client using the Observer Pattern.
* “error” : if a validation test is not passed, the Error Objects gets enabled and a message is set up.
* “return value”: the network protocol defined requires that normal actions (and the update messages coming from the model manipulation) are followed by a MSG\_END message. The Thread calling the action applies that final message, if any action returns true. It does not otherwise.

ACTIONS BEHAVIOR:

1. chooseLeaderCards(…)

The Client is shown 4 different leadercards, because it has a LeaderCardsMiddleObject enabled. When he finally chooses 2 of them, the actionManager gets activated. The message contains the two leaderCards. The controller associates the cards in the message with the actual player. The powerful endTurn() may skip inactive players, may change the cards in the object, may disable the middle-object, all depending on the status of the model.

1. chooseResource(…)

In a very similar manner as the (1) action, the message contains the selected resource. It is then added to the player depot. The EndTurn is then called.

1. ActivateLeaderCard(…)

The message contains the number of the card that the player wants to activate. After checking that the card is not active (model validation), it is then checked if the player possesses all the necessary requirements, and if so, the card gets enabled.

1. DiscardLeaderCard(…)

Basically same as (3) with destructing intents. And the player advances, too.

1. activateProduction(…)

The message contains all the necessary informations:

* 1. IF the player wants to activate the basic production and IF SO, the input and output
  2. IF the player wants the standard production, and IF SO, the card numbers
  3. IF the player wants to activate the leader production, and IF SO, the card numbers

It is then checked if the player possesses all the necessary resources. If so, the output is thrown in his strongbox.

1. changeDepotConfig(…)

The message contains a new configuration for the warehouse depot and the extradepot. It is checked if the number of resources are the same, and in a correct disposition. If so, that new configuration is applied.

1. buyDevelopmentCard(…)

The controller makes a list of possible cards that the player could buy. The list (in reality a map), contains information as: <Card, possible slots to put the card in>. If the player can’t buy any card, the error is set. Otherwise, the vendor object is set to enable with that information stored. It is then mirrored to the client, which could…

1. chooseDevelopmentCard(…)

The player has received the list of the vendor. It may choose a card, or it may not choose a card. The “not choose” is specified by a -1,-1 value contained in the message. If the values are not -1,-1, then the player has chosen to buy a card (which one, value#1 of the message), and where to put it (slot position, value#2 of the message). The controller \*only performs a validation of the message/vendorObject\*, cause the proper requirements validations has already been performed in the (7) action. If everything goes well, the new card pops in in the player’s development slot, after he paid the price.

1. getMarketResources(…)

The player selects a row or a column. The message contains the row/column Boolean value, and the actual row/column numeric value. The Controller pushes the market row/column and gets a list of resources. The white marble may be automatically converted, if the player has only one leader card with a MarketAbility and it is enabled. Otherwise (no cards) it is scrapped, or (2 cards enabled) he may choose in which resource to transform it.

The marketHelper gets enabled. The MarketHelper contains a list of possible choices to do with a “current selected resource”:

Throw it in the deposit, throw it in the extradepot, discard it, swap his depot rows, convert it..

The choices are updated when the player calls the:

1. newChoiceMarket(…)

Basically it is an interaction with the player, which is choosing what to do with a certain resource grabbed from the market. The message contains a number, corresponding to a choice. The choice, if possible, translates in a determined action (see above). At the end, when the player has dismissed all its resources, the MarketHelper is set to false.

1. endTurn(…)

In the initial phases (1 or 2), the endTurn is called by the chooseLeaderCards and chooseResource methods. It will be called if the currentPlayer crashed. The endTurn contains the logic behind the disconnection, the reconnection, the skipping idle players, manipulating middle-objects in the initial phases of the game, the status changes, and resets the permitted action.

Note: the permitted action. The controller requires that if a player has performed positively (5), (7) or (9), he cannot do the anymore. This check is performed both on server side, and on client side. If a player has performed a “main move” or “powerful action”, he simply can’t do more of them. He could still perform basic actions, like changing depot configuration, discarding/enabling leaderCards. The permitted action gets resetted when he performs the end Turn action.