A player chooses to activate a Leader Card: explanation

During their turn, the user chooses to play a leader action, specifically to activate a leader card.

The method AskForLeaderCardToActivate() is called inside the AskForNextAction() function.

It generates a form which displays the possible leader cards to activate and returns a new ActivateLeaderCardEvent(), which is returned to the Network Handler.

The Network Handler sends the Event to the Virtual View (by conversion to Json).

The Virtual View sends the Event to the Controller, whose handler ActivateLeaderCardEventHandler() invokes the activateLeaderCard() method on the LeaderCardManager.

The LeaderCardManager.activateLeaderCard() modifies the model, specifically calling on the Player instance the activateLeaderCard() function, which modifies the states of the specified LeaderCard .

If the operation is legal, it ends successfully, therefore the changes in the model state are notified to the controller, which generates a new Event: PlayerStateEvent(), which contains the updated hashmap with the player's new leader cards.

Along with that, the ActivateLeaderCardEventHandler() of the controller, also changes the turn state in the matchState instance, and also this update of the model is notified and generates a new MatchStateEvent() in the controller.

These new Events(PlayerStateEvent and MatchStateEvent) are sent to the Network Handler from the Virtual View, and are answered respectively by the PlayerStateEventHandler and the MatchStateEventHandler in the Network Handler. The former calls the AskForNextAction() method in the View, the latter calls the updateLeaderCardsState() in the View.

However, if the operation in the model is not valid or is unsuccessful, some error Events are generated , these are the cases:

1. If requirements are not met, a RequirementsNotMetException arises and , when caught , sends a new RequirementNotMetError to the Network Handler.
2. If the card is already active, an IllegalOperation arises and if caught sends a PlayerActionError to the Network Handler.
3. If the player doesn't own the leader card in the model, the state is inconsistent and generates a NotPresentException, which, when caught , sends a BadRequestEvent to the Network Handler.

For the first two cases, the view visualizes a message error and the controller restores the state of the match sending a new MatchStateEvent() to the Network Handler; for the third case, instead, the program terminates, so it should never be the case it happens.