Commands Overview

This section provides an overview of the various commands used in the game. Commands are created by the view based on the client's input, sent to the server, and executed within the game's controller state.

Command List

- **ChatCommand**
  - Description: Updates the chat with the latest messages from players.
  - Execution: Calls `controller.updateChat(clientId, message, sender, receiver)`
- **ChooseCommand**
  - Description: Allows players to choose the initial card's side and which objective card to use.
  - Execution: Calls `controller.chooseSetUp(clientId, player, side, objCard)`
- **CreateGameCommand**
  - Description: Initializes a new game. This command sets up a new game instance, creating the first player and initializing all the necessary game components.
  - Execution: Calls `controller.initialized(clientId, nickName, password, color, numOfPlayers)`
- **DrawCommand**
  - Description: Handles the drawing of cards by a player. It updates the game state based on the drawn card and manages the turn transition.
  - Execution: Calls `controller.drawnCard(clientId, player, card, fromDeck)`
- **JoinGameCommand**
  - Description: Manages the process of a player joining an existing game. It ensures that the player credentials are valid and adds the player to the game.
  - Execution: Calls `controller.joinGame(clientId, nickName, password, color)`

- **Ping**
  - Description: A simple command used to check the connectivity between the client and the server. The server has a timeout for the ping command to ensure the client is still connected. If the client does not respond within the timeout, the server will disconnect the client.
- **PlaceCommand**
  - Description: Handles the placement of cards by a player. It ensures the card is placed according to the game rules and updates the game state accordingly.
  - Execution: Calls `controller.placedCard(clientId, player, father, placeThis, position, frontUp)`
- **RejoinGameCommand**
  - Description: Allows a previously disconnected player to rejoin the game. It verifies the player's identity and reconnects them to the game.
  - Execution: Calls `controller.rejoinGame(clientId, nickName, password)`

Events Overview

This section provides an overview of the various events used in the game. Events are created by the model to communicate its changes to the clients. They are passed to the server and sent to the clients through the network, where the clients call the `doJob` method, passing the `MiniModel`. The `MiniModel` is then responsible for updating the client's view based on the event.

Event List

- **ChatEvent**
  - Description: Updates the chat with the latest messages from players.
  - Execution: Calls `miniModel.setChat(chat)`
- **ChooseEvent**
  - Description: Updates the game state based on the player's initial choice of the card's side and the objective card.
  - Execution: Calls `miniModel.setState(state)`, `miniModel.setHands(hands)`, and `miniModel.setPlayers(players)`
- **DrawEvent**
  - Description: Updates the game state after a player draws a card. It updates the hands, current player, deck, and board.

  - Execution: Calls `miniModel.setBoard(board)`, `miniModel.setHands(hands)`, `miniModel.setDeck(deck)`, `miniModel.setCurrentPlayer(currentPlayer)`, `miniModel.setState(state)`, `miniModel.setTurnCounter(turnCounter)` and `miniModel.setLastTurn(lastTurn)`

- **EndGameEvent**
  - Description: Signals the end of the game and provides the final scores.
  - Execution: Calls `miniModel.setBoard(board)`, `miniModel.setState(state)`, and
    `miniModel.setWinners(winners)`

- **ErrorEvent**
  - Description: Communicates an error that occurred during the game.
  - Execution: Calls `miniModel.setError(error)`

- **ForcedEndEvent**
  - Description: Signals the forced end of the game due to player disconnection.
  - Execution: Calls `miniModel.setError(cause)`

- **InLobbyEvent**
  - Description: Indicates that the player is in the lobby waiting for the game to start.
  - Execution: Calls `miniModel.setGameId(gameId)`, `miniModel.setNickName(nickName)`, and `miniModel.setState(state)`

- **PlaceEvent**
  - Description: Updates the game state after a player places a card. It updates the structures, hands, and board.
  - Execution: Calls `miniModel.setPlayerStructure(playerStructure)`, `miniModel.setHands(hands)`, `miniModel.setBoard(board)`
    and `miniModel.setState(state)`

- **RejoinGameEvent**
  - Description: Allows a previously disconnected player to rejoin the game. The model sends to the client the current state.
  - Execution: Calls `miniModel.setGameId(gameId)`, `miniModel.setNickName(nickname)`, `miniModel.setPlayers(players)`, `miniModel.setPlayerStructure(playerStructure)`, `miniModel.setHands(hands)`, `miniModel.setBoard(board)`, `miniModel.setDeck(deck)`, `miniModel.setCurrentPlayer(currentPlayer)`, `miniModel.setNextPlayer(nextPlayer)` and `miniModel.setState(state)`

- **StartGameEvent**
  - Description: Signals the start of the game and provides the initial game state.
  - Execution: Calls `miniModel.setPlayers(players)`, `miniModel.setPlayerStructure(player` `miniModel.setHands(hands)`, `miniModel.setBoard(board)`, `miniModel.setDeck(deck)`, `miniModel.setCurrentPlayer(currentPlayer)`, `miniModel.setNextPlayer(nextPlayer)` and `miniModel.setState(state)`
- **WaitEvent**
  - Description: Indicates that the player should wait for other players.
  - Execution: Calls `miniModel.setGameId(gameId)` and `miniModel.setState(state)`