Sequence Diagrams

Gruppo PSP38

Elisabetta Fedele
Elia Fantini
Raffaele Berzoini

Contents

1 Player access to the game

1.1 New player protocol

After the server has been started with its own port number, a client can start a connection by providing the server's port number and IP address or using the default configuration.
After having accepted the request, the server asks the game mode the client wants to play (solo game or multiplayer game) with a GameModeRequest.
The client then reply with a GameModeResponse, which contains the chosen game type. The client itself before sending the message checks its validity.
In both the cases, the server asks the client to provide a syntactically valid nickname (NicknameRequest), with characters allowed A-Z, a-z and 0-9. The client replies with a NicknameResponse containing the chosen nickname. The server checks if the nickname is unique in the system and, if it is not, a NicknameRequest(true, true) is sent to that client. If the nickname has to finish an old game in the game mode he has previously selected, he is reconnected to the room of that game if other players are still playing. If the game has finished while the client was disconnected, the server will just send to him the results of that game and will reconnect him to the main lobby.

When the server receives a valid username and the client has no previous game to finish, the server reacts in two different ways, according to the game mode previously chosen.

1.1.1 `MULTIPLAYER` game mode

1. The nickname is saved in the ClientHandler
2. Once the nickname has been saved, the ClientHandler of the specific connection is added to a linked queue of connections in the Server and the server invokes the newGameManager() method
3. The workflow of newGameManager() depends on the status of the queue of players:
   - First case: the server has not already asked the number of players desired to the first client of the queue
     - The server sends a NumberOfPlayersRequest to the client with isRetry set as false.
     - The client sends his desires with a NumberOfPlayersResponse. His response is be checked on the client side, so the server receives only valid responses
     - After having received a response, newGameManager will be called another time
   - Second case: the number of players of the next match is already known and the lobby contains enough players (<=NumberOfPlayersForNextGame) to start a new game
     - The server checks whether there is any invalid nickname among the players ready to play
     - If there isn't, a new Game is created and the players of that game are removed from the queue.
     - When a new name is sent by the client, the method newGameManager() is called again. As a consequence, the check goes on until all the nicknames of the first NumberOfPlayerForNextGame players are valid.
   - In all the other cases the method returns and everything remains unchanged.

1.1.2 `SOLO` game mode

The nickname is saved in the connection and a new Solo Game is created

1.2 Lobby phase messages

1.2.1 Messages to client

- GameModeRequest()
- NicknameRequest(boolean isRetry, boolean alreadyTaken)
- NumberOfPlayersRequest(boolean isRetry)
- WaitingInTheLobbyMessage()
- SendPlayersNicknamesMessage(String playerNickname, List<String>otherPlayersNicknames)

1.2.2 Messages to server

- GameModeResponse(GameMode gameMode)
- NicknameResponse(String nickname)
- NumberOfPlayersResponse(int numberOfPlayers)

1.3 Sequence diagram

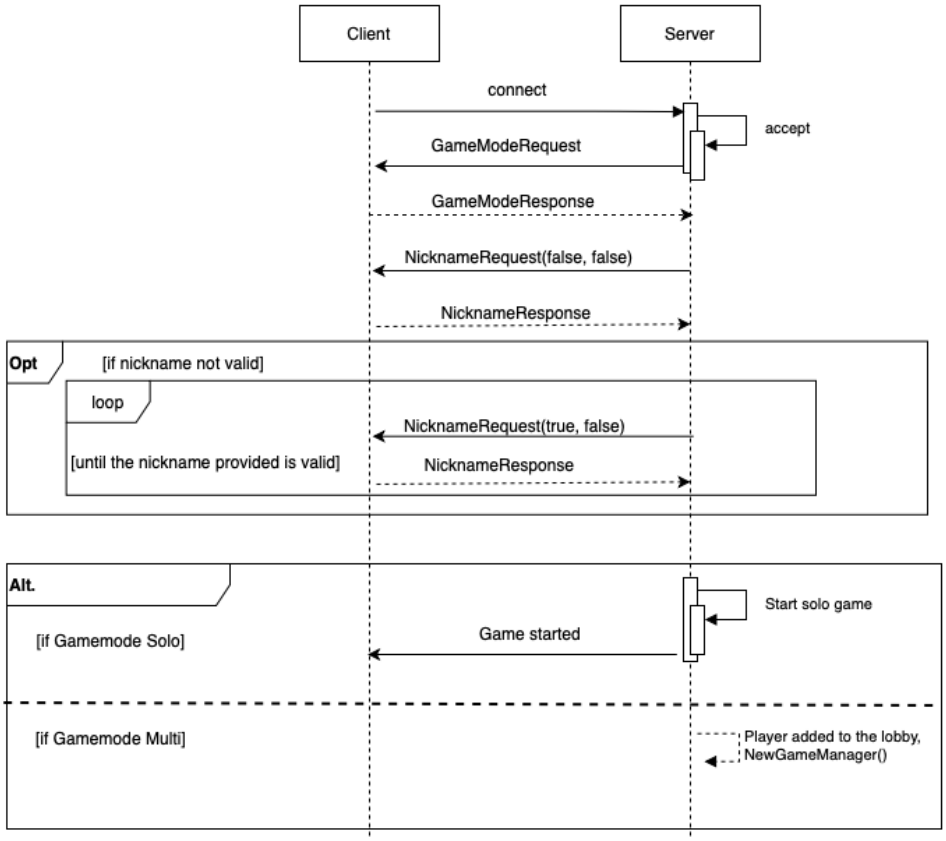

Figure 1: Player's access to the server

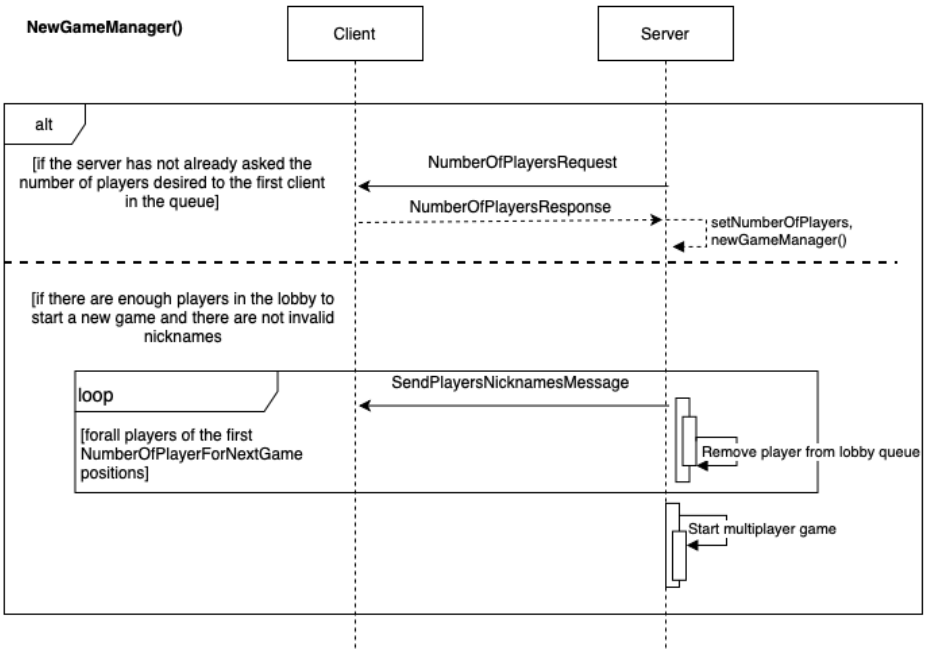

Figure 2: NewGameManager's workflow

2 Play a leader action

2.1 `PLAY_LEADER_CARD_ACTION` protocol

1. First, the server sends a DoActionRequest to the client, where he displays all the possible actions that can be done by the client
2. If a `PLAY_LEADER_CARD_ACTION` is available, the client can sends a DoActionResponse containing as ActionType `PLAY_LEADER_CARD_ACTION`.
3. The server sends to the client a ChooseLeaderCardToPlayRequest containing a list of his inactive leader card, asking the client to choose which one he wants to activate.
4. The client replies with a ChooseLeaderCardToPlayResponse containing the id of the selected card chosen from a list of the available ids. The check of the correctness of the response is made on client side, so the server will always receive a valid response.
5. The controller activates the selected leader card
6. After the action is finished, the server checks whether the client has other available actions:
   - If the client still has some available action to complete his turn, the Server send to him a DoActionRequest (which includes the possibility to end the turn only if the mandatory standard action has already been done before)
   - If the client cannot do any action, the server notifies him of the end of his turn with a EndTurnMessage with isForced = true

2.2 `PLAY_LEADER_CARD_ACTION` messages

2.2.1 Messages to client

- ChooseActionRequest(Map<ActionType, boolean>availablesActions, boolean standardActionDone)
- SelectCardRequest(List<Integer>inactiveLeaderCardIds, boolean leaderOrDevelopment)
- TurnMessage(String nickname, boolean isStarted)

2.2.2 Messages to server

- ChooseActionResponse(int actionChosen)
- SelectCardResponse(Integer id)

2.3 Sequence diagram

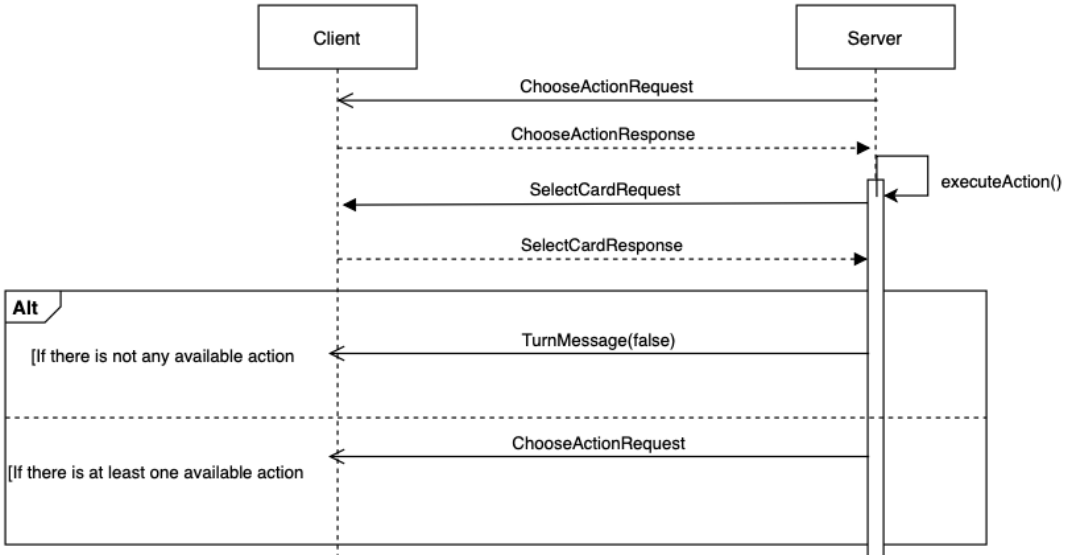

Figure 3: Play leader card action's flow

3 Take Resource From Market Action

3.1 `TAKE_RESOURCE_FROM_MARKET_ACTION` protocol

1. First, the server sends a ChooseActionRequest to the client, where he displays all the possible actions that can be done by the client
2. If a `TAKE_RESOURCE_FROM_MARKET_ACTION` is available (it could not be available since only one standard action for turn is allowed), the client can sends a ChooseActionResponse containing as ActionType `TAKE_RESOURCE_FROM_MARKET_ACTION`
3. Then, the server asks to the client the insertion position of the marble, with a MarbleInsertionPositionRequest
4. The client chooses a position and sends it back to the server with a MarbleInsertionPositionResponse
5. The correctness of the response is verified on the client's side, so the server will receive only valid responses
6. The marble is then inserted in the market grid and a list of 4 marbles to convert is given to the player:
   - If this list contains at least one white marble and if the player has got more than one leader card with `WHITE_MARBLE_EFFECT` active on his Leader Board [**condition 1**], for every white marble, the server will ask to the client how he desires to convert it with a ChooseWhiteMarbleConversionRequest, showing the two available conversions. The client will reply with a ChooseWhiteMarbleConversionResponse.
   - If the client has taken white marble(s), but he only possess one active leader card with `WHITE_MARBLE_EFFECT`, the conversion of the marble(s) will be made automatically, since once an effect is active, the client has no choice but to use it.
   - If no white marble has been taken, the marbles remain unchanged.
7. Then, the marbles will be converted in resources (or in faith points) according to the game rules.
8. For each converted resource, the server will ask to the client where he desires to store it with a ChooseStorageTypeRequest (List<PossibleStorageMode>), providing a list of the available storages. If the list is empty the client can decide whether to reorganize his depots or to discard the resource. The discard of the resource is always possible by setting true the boolean discard in the response
9. The client will then decide where to store the resource. If the client decides to discard a resource, it will be handled by the controller in two different ways in the solo and in the multiplayer game mode, by using a strategy pattern
10. When all the resources of the list have been stored, the server will check whether the client has other available actions:
    - If the client still has some available action to complete his turn, the Server send to him a ChooseActionRequest (which also includes the possibility to end the turn, since the mandatory standard action has already been done)
    - If the client cannot do any action, the server notifies him of the end of his turn with a TurnMessage

3.2 `TAKE_RESOURCE_FROM_MARKET_ACTION` messages

3.2.1 Messages to client

- ChooseActionRequest(Map<ActionType, boolean>availablesActions, boolean standardActionDone)
- MarbleInsertionPositionRequest()
- ChooseWhiteMarbleConversionRequest(List<Resource>, int numberOfMarbles)
- ChooseStorageTypeRequest (List<PossibleDepot>availablesDepots)
- EndTurnMessage(boolean isForced)

3.2.2 Messages to server

- ChooseActionResponse(int actionChosen)
- MarbleInsertionPositionResponse(int position)
- ChooseWhiteMarbleConversionResponse (List<Resource>conversionsChosen)
- ChooseStorageTypeResponse (Resource resource, String storageType, boolean canDiscard, boolean canReorganize)

Where

- PossibleDepot = `WAREHOUSE_FIRST_ROW`, `WAREHOUSE_SECOND_ROW`, `WAREHOUSE_THIRD_ROW`, `LEADER_SERVANT`, `LEADER_COIN`, `LEADER_SHIELD`, `LEADER_STONE`
- ActionType = `DISCARD_LEADER`, `PLAY_LEADER`, `TAKE_RESOURCES_FROM_MARKET`, `ACTIVATE_PRODUCTION`, `BUY_DEVELOPMENT_CARD`

3.3 Sequence diagram

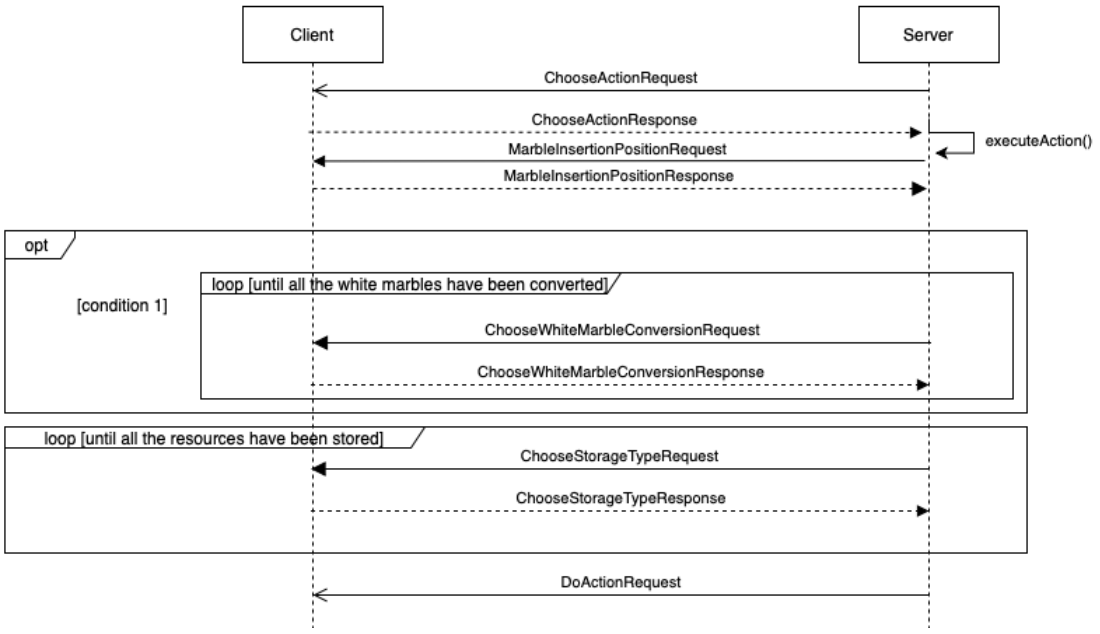

Figure 4: Take resources from market action's flow

4 Reorganize depots

4.1 Reorganize depots protocol

After having received a ChooseStorageTypeRequest message, the player has the opportunity to reorganize his depots using the SwapWarehouseDepotsRequest and the MoveResourcesRequest. If the server receives a

- SwapWarehouseDepotsRequest which includes the two depots to be swapped
  1. If the swap is possible, the controller swaps the depots, otherwise he does not and notify the client of the failure
  2. In both the situations he sends a sendReorganizeDepotsCommand back to the client, with failure = false only if the swap was possible and has been done correctly
- MoveResourcesRequest which includes: origin depot, destination depot, number of resources to be moved
  1. If the move is possible, the controller moves the resources, otherwise he does not
  2. In both the situation he send a SendReorganizeDepotsCommand back to the client, with failure = false only if the move was possible

4.2 Reorganize depots messages

4.2.1 Messages to client

- SendReorganizeDepotsCommands(List<String>availableDepots, boolean first, boolean failure, List<Resource>availableLeaderResources)

4.2.2 Messages to server

- SwapWarehouseDepotsRequest(String originDepot, String destinationDepot)
- MoveResourcesRequest(String originDepot, String destinationDepot, Resource resource, int quantity)

4.3 Sequence diagram

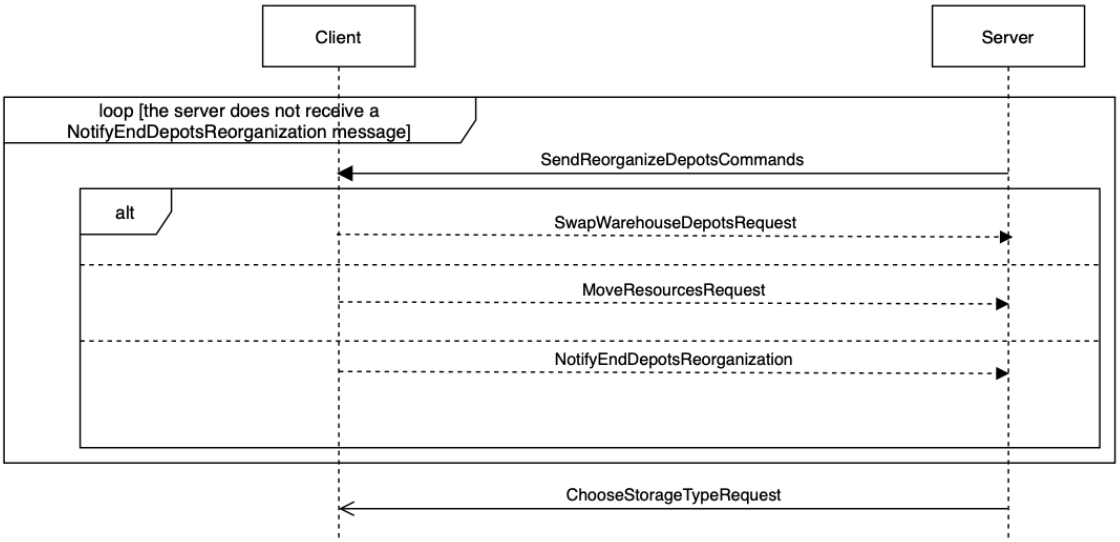

Figure 5: Reorganize depots flow