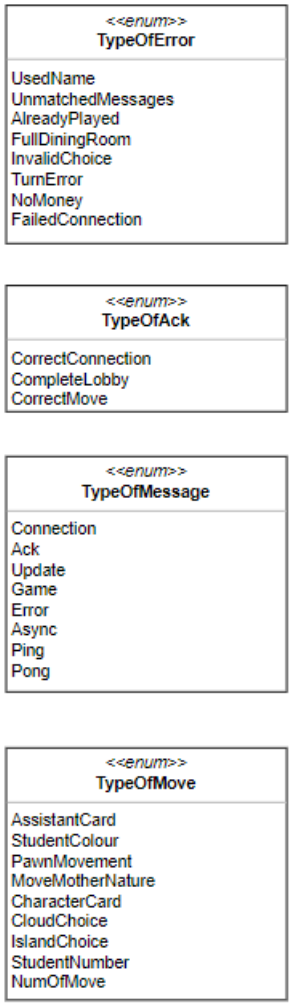
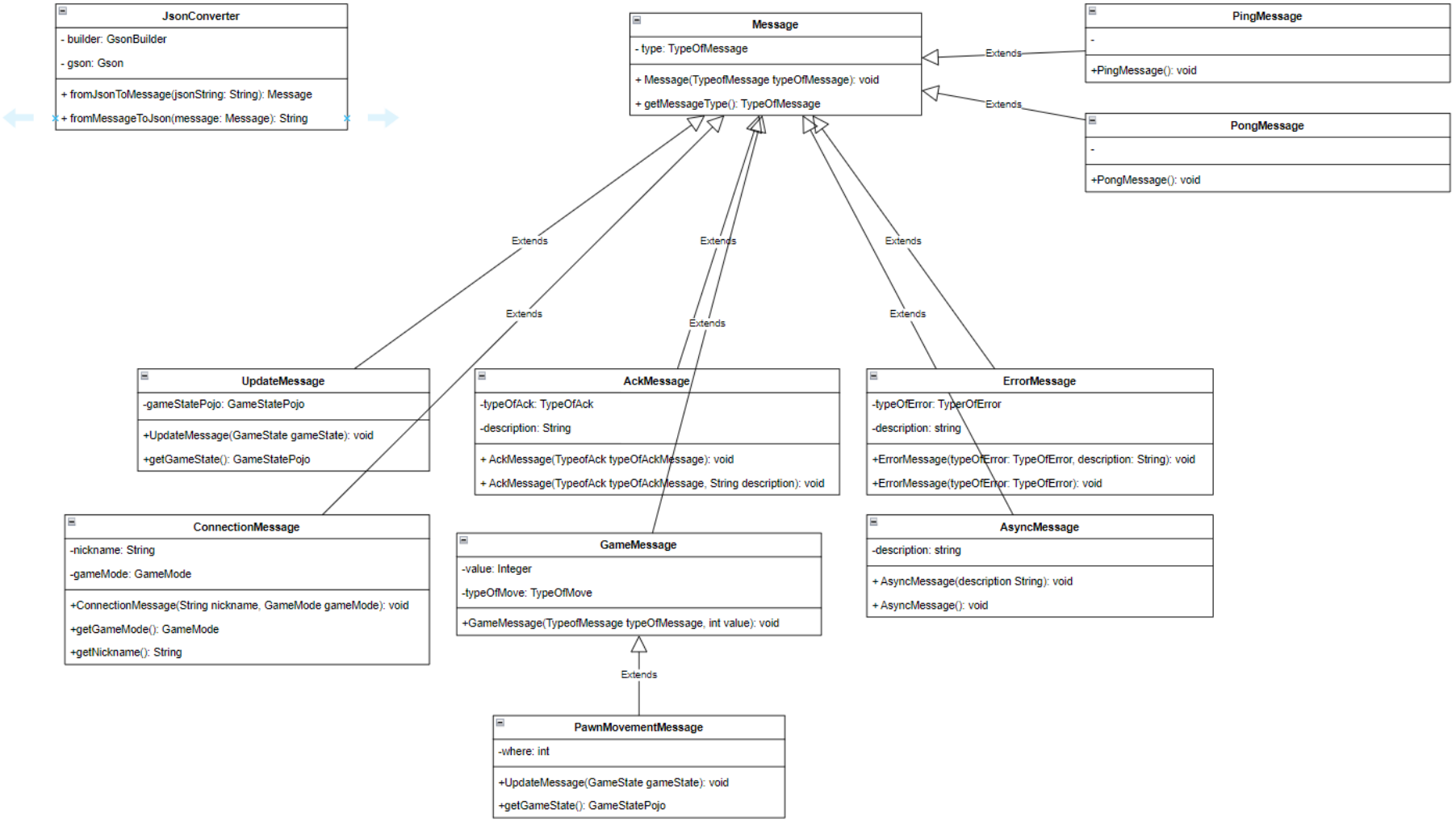
Eryantis Protocol Documentation

Lorenzo Franzè, Valentina Moretti, Lara Nonino

Gruppo 22

Messages: UML diagram




JsonConverter
- builder: GsonBuilder
- gson: Gson
+ fromJsonToMessage(jsonString: String): Message
+ fromMessageToJson(message: Message): String
Message
- type: TypeOfMessage
+ Message(TypeofMessage typeOfMessage): void
+ getMessageType(): TypeOfMessage
PingMessage
-
+PingMessage(): void
PongMessage
-
+PongMessage(): void
Extends
UpdateMessage
-gameStatePojo: GameStatePojo
+UpdateMessage(GameState gameState): void
+getGameState(): GameStatePojo
AckMessage
-typeOfAck: TypeOfAck
-description: String
+ AckMessage(TypeofAck typeOfAckMessage): void
+ AckMessage(TypeofAck typeOfAckMessage, String description): void
ErrorMessage
-typeOfError: TyperOfError
-description: string
+ErrorMessage(typeOfError: TypeOfError, description: String): void
+ErrorMessage(typeOfError: TypeOfError): void
ConnectionMessage
-nickname: String
-gameMode: GameMode
+ConnectionMessage(String nickname, GameMode gameMode): void
+getGameMode(): GameMode
+getNickname(): String
GameMessage
-value: Integer
-typeOfMove: TypeOfMove
+GameMessage(TypeofMessage typeOfMessage, int value): void
AsyncMessage
-description: string
+ AsyncMessage(description String): void
+ AsyncMessage(): void
PawnMovementMessage
-where: int
+UpdateMessage(GameState gameState): void
+getGameState(): GameStatePojo

Message

General message from which each kind of message inherit. It is used when the only need is to explicit the TypeOfMessage without further information.
In our code we check the typeOfMessage when needed, and from this information we create the message using the correct class.

Arguments

- TypeOfMessage: enum that describes the content of message.

  It can be Connection(0), Ack(1), Update(2), Game(3), Error(4), Ping(5), Async(6).

Possible Responses

- All kind of messages.

ConnectionMessage

Message sent from Client to Server when the former wants to establish a connection to create a game session. The message is sent only after some local checks on the client: the nickname must be at least of 4-char length and the game mode must be only on the suggested range (from 1 to 4 as the possible alternatives are: simple game mode with 2 or 3 players and expert game mode with 2 or 3 players). The server will check the nickname univocity: if the nickname is already in use the response will be an “ErrorMessage”, otherwise it will be an “AckMessage” and the player is correctly admitted in the waiting-lobby. As simultaneous game are permitted, there are always 4 waiting lobbies opened. The new player is admitted in the one he prefers (information picked from the GameMode attribute). When the lobby reaches the correct number of players according to the GameMode it is associated with, the lobby is drained and the players start the game.

Arguments

- TypeOfMessage: Connection.
- Nickname: specifies the name of the player who wants to establish a connection.
- GameMode: specifies the game mode chosen.

Possible Responses

- ErrorMessage: the nickname is already used.

- AckMessage: connection accepted.

GameMessage

Message sent from the client to the server to communicate the type of move which is specified through the attribute TypeOfMessage and its value, specified through the attribute Value.

Arguments

- TypeOfMessage: enum that describes the type of move.

  It can be AssistantCard(0), StudentColour(1), StudentMovement(2), MoveMotherNature(3), CharacterCard(4),CloudChoice(5), IslandChoice(6), StudentNumber(7).

- value: value of the game move. Eg: the number of steps the player wants to move the mother of nature.

Possible Responses

- AckMessage: the game move is valid and accepted. This message is followed by the UpdateMessage to update the client's view.

- UpdateMessage: client's choose is right; this message updates client's view.

- ErrorMessage: the game move is invalid and refused and through the TypeOfError we can display the reason of the error. The player must take another game move choice if it is his turn and the game move is necessary (it is not the case of an invalid character card choice because it is not required to continue the game).

PawnMovementMessage

PawnMovementMessage inherits from GameMessage. It is used when the choice of the players move requires two attributes to be communicated. For example to move a student the client controller can send in an unique message both the information on the color and of the number of students to move. It is also useful for some of the character cards.

Arguments

- TypeOfMessage: enum that describes the type of move.

  It can be AssistantCard(0), StudentColour(1), StudentMovement(2), MoveMotherNature(3), CharacterCard(4),CloudChoice(5), IslandChoice(6),

StudentNumber(7);

- valueA: the first value of the game move.
- valueB: the second value of the game move.

Possible Responses

- AckMessage: the game move is valid and accepted. This message is followed by the UpdateMessage to update the client's view.
- UpdateMessage: client's choose is right; this message updates client's view.
- ErrorMessage: the game move is invalid and refused and through the TypeOfError we can display the reason of the error. The player must take another game move choice.

UpdateMessage

Message sent from the server to the client to communicate the current game state and trigger an update of the view.

Arguments

- TypeOfMessage: Update.
- GameState: contains the game state.

Possible Responses

No response is needed

AsyncMessage

Message sent from the client to the server to communicate the client has disconnected or to the server to all the remaining clients to inform about the disconnection of the server or of another player. A client get disconnected when the ping sent from the server doesn't come back before the timeout, or when there is an error with his socket, or when his socket is still connected but he is taking too much time to make his move-choice.

Arguments

- TypeOfMessage: Async.

Possible Responses

- Another AsyncMessage is sent to the other players.
- No responses to the disconnected-player as he has just disconnected.

PingMessage

Message sent from the client to the server and viceversa to check the disconnections.
When it is received a Boolean variable is set to true. This variable is checked after a timeout. If it value is true, it means the connection is still alive. The process is repeated again: the variable is set to false and another timeout start.

Possible Responses

- A PongMessage.

PongMessage

Message sent from the client to the server and viceversa to check the disconnections.
It is sent when it is received a PingMessage.

AckMessage

Message sent from the server to the client to inform about the correct move-choice or selection.

Arguments

- TypeOfAck: enum that describes the type of ack.

  It can be CorrectConnection(0), CompleteLobby(1), CorrectMove(2);

- description: to display the connected string message according to the ack received.

Possible Responses

No response is needed.

ErrorMessage

Message sent from the server to the client to inform about an error that has occurred. In the connection-phase it can be used if the player has chosen an invalid nickname. During the pianification and action phases there can be different kind of errors according to the move chosen by the client.

Arguments

- TypeOfAck: enum that describes the type of error occurred.

  It can be UsedName, UnmatchedMessages, FullDiningRoom, InvalidChoice, TurnError, NoMoney, FailedConnection.

- description: to display the connected string message according to the error received.

Possible Responses

- GameMessage: to ask the player to re-choose a valid value for this move.

Scenarios

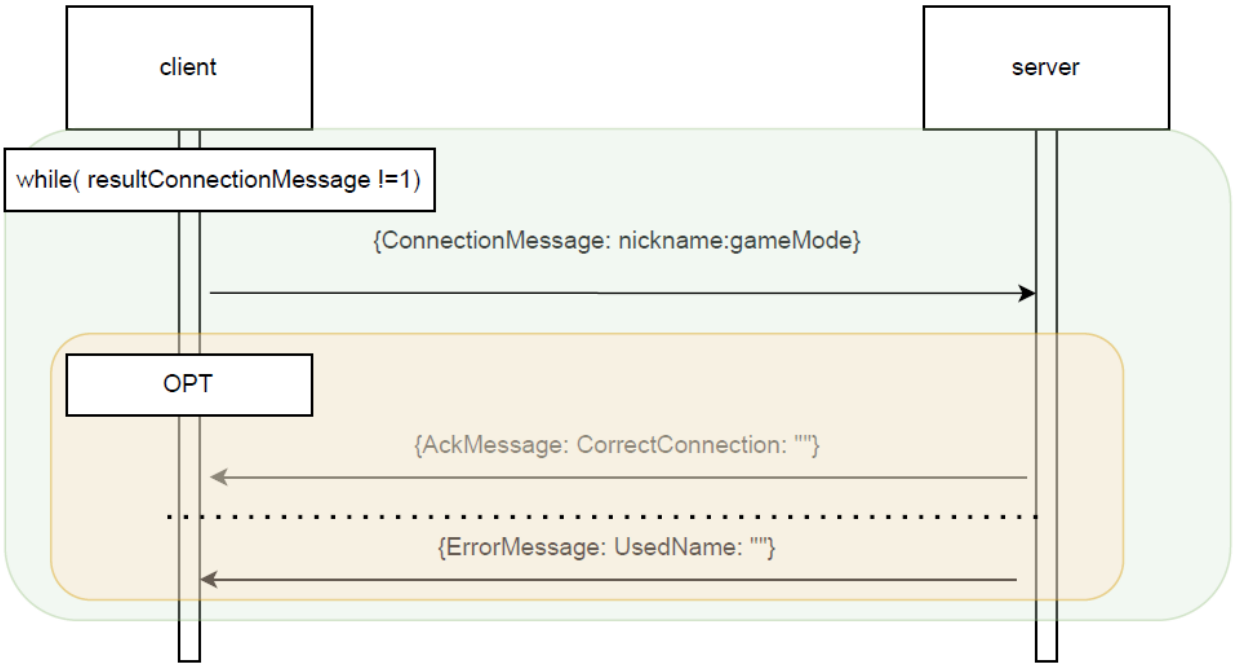
Connection

After the connection is established with an “accept()” by the server, the server handles the connection of the player.
The client should send a ConnectionMessage in witch he communicates the nickname he wants to play with and the gameMode he wants to play.

There are four types of GameMode:
- Simple with 2 players
- Simple with 3 players
- Complex with 2 players
- Complex with 3 players

The server checks the nickname univocity (all the clients connected must have different nicknames) and assign the client to the correct lobby (chosen according to the GameMode).
If the nickname- univocity test is passed, the server accepts the nickname and informs the client (with an “Ack” response).

If another player is already using that nickname, the server refuses the nickname and inform the client (with an “Error” response).



Phases:

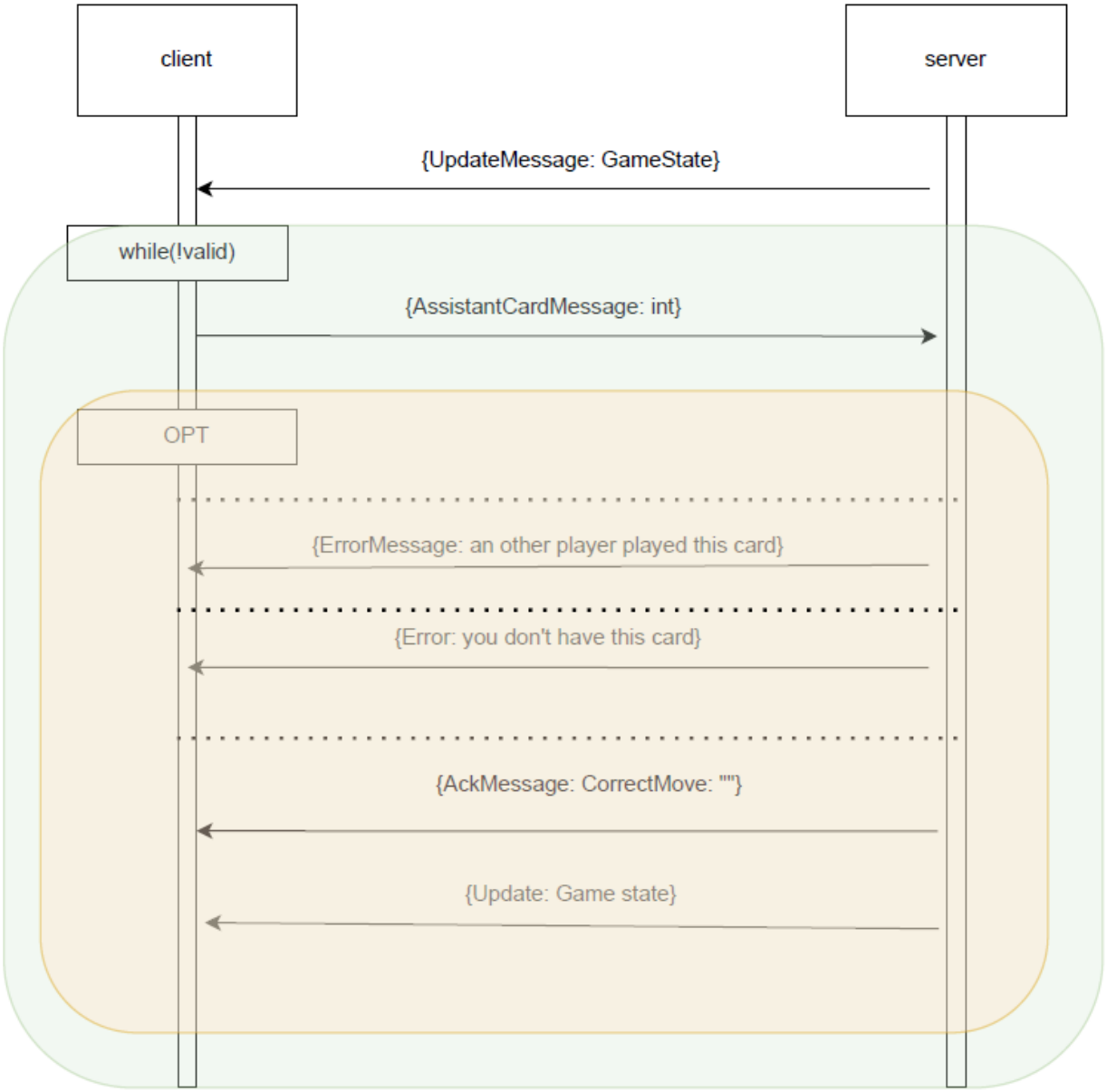
The GameController has three phases: SetUpPhase, PianificationPhase and ActionPhase.
The first one (SetUpPhase) doesn't requires any interaction between client and server. For this reason the following sequence diagram are about the other two phases.

PianificationPhase:

In the pianification phase just one choice of the client is required: the one of the assistant card. After an update message, the client sends his choice on the card. Some primary checks are made by the clientController.

Also the server controller has the job to check if the chosen card is valid. It may happen that it is an invalid move because the client doesn't have that card or because another player has chosen the same card (and that card is not the last one in the hands of the client).

In these cases, an error message is sent by the server and the process start again from a new choice by the client. Otherwise, an ack message and an update message are sent to all the clients and the action phase can take place.


client
server
{UpdateMessage: GameState}
while(!valid)
{AssistantCardMessage: int}
OPT
{ErrorMessage: an other player played this card}
{Error: you don't have this card}
{AckMessage: CorrectMove: ""}
{Update: Game state}

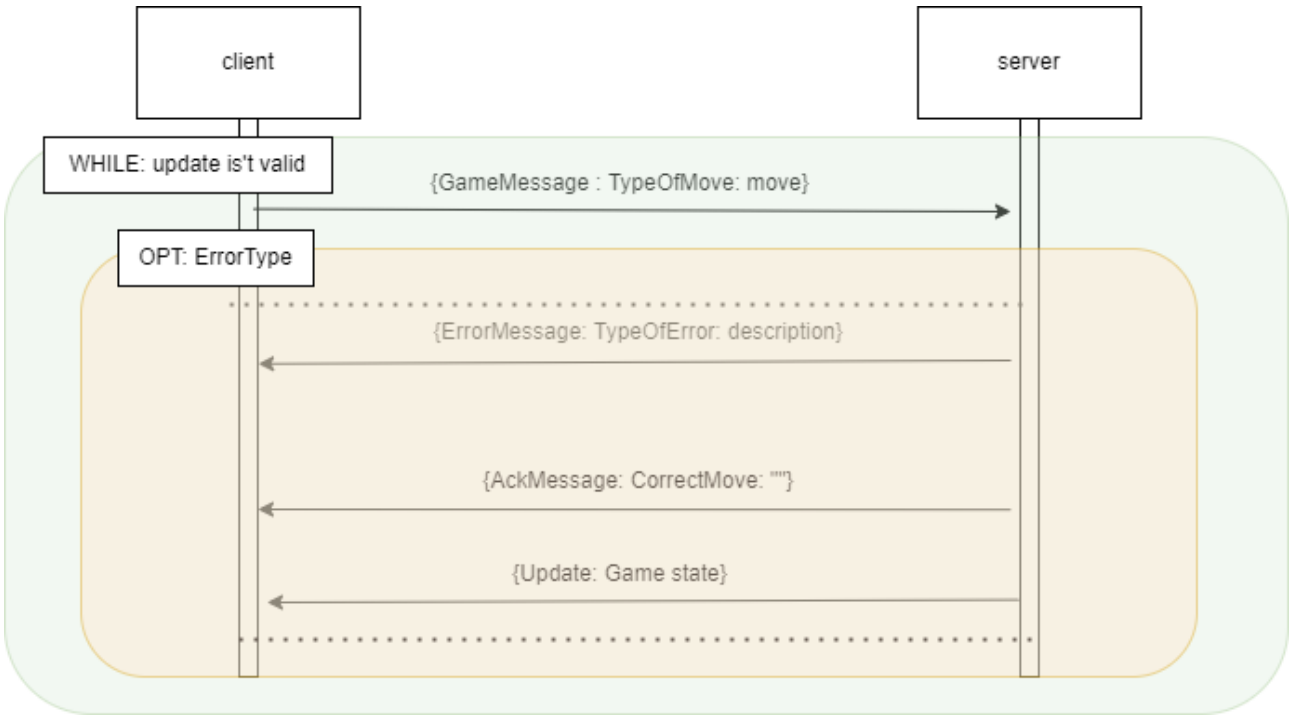
ActionPhase:

According to the message sent from the client, there will be two scenarios (the client can send a GameMove or a CharacterCard).

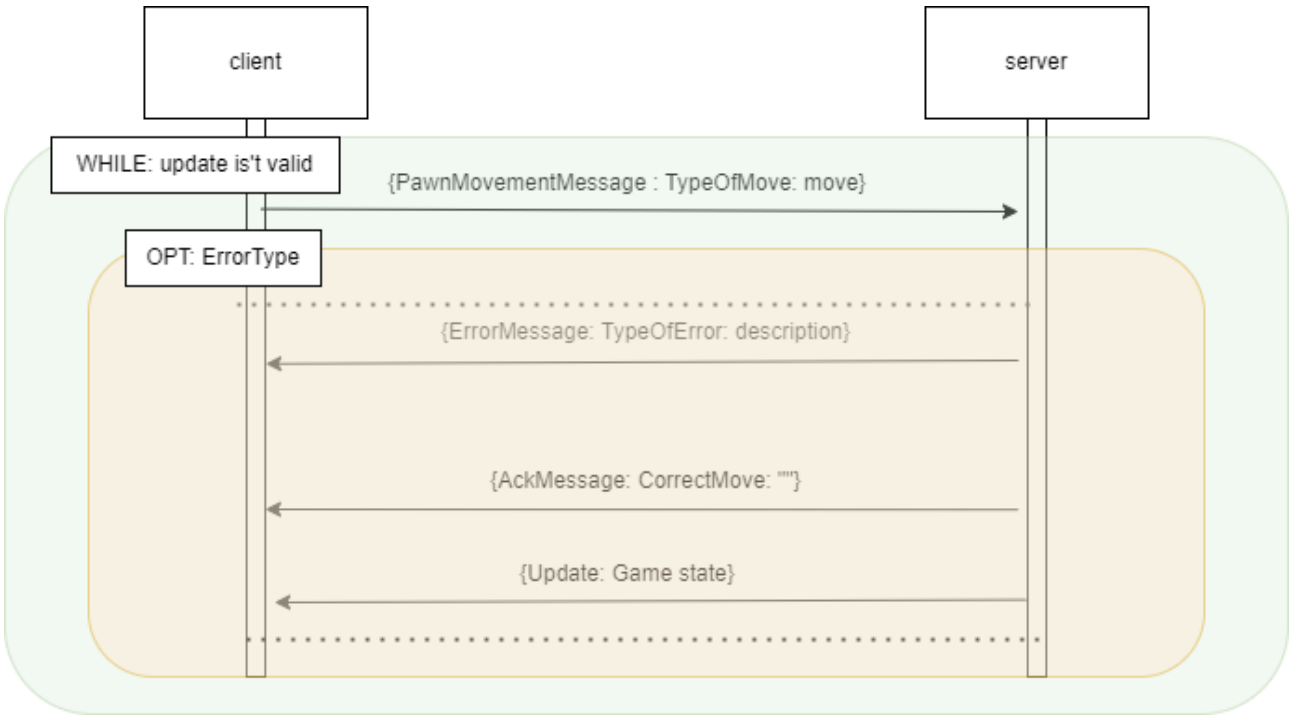
The action phase has a three-phase schedule: first the choice of the students to move (messages of the student's color and his new position are sent by the client), then the choice of mother nature's steps, then the cloud (the last player receives the remaining one without being able to choose it). The sequence diagrams of these steps are the same. This is because the message that are sent are all of the same class: GameMessage.

The only difference is found in the "messageType" declared in the GameMessage's attribute. Here the type "messageType" used in this phase: Connection(0), Ack(1), Update(2), Game(3), Error(4), Ping(5), Async(6).

Because of the regular scheme in witch these three choice are made, we have represented a generic sequence diagram that is followed in each of the three steps of the action phase.

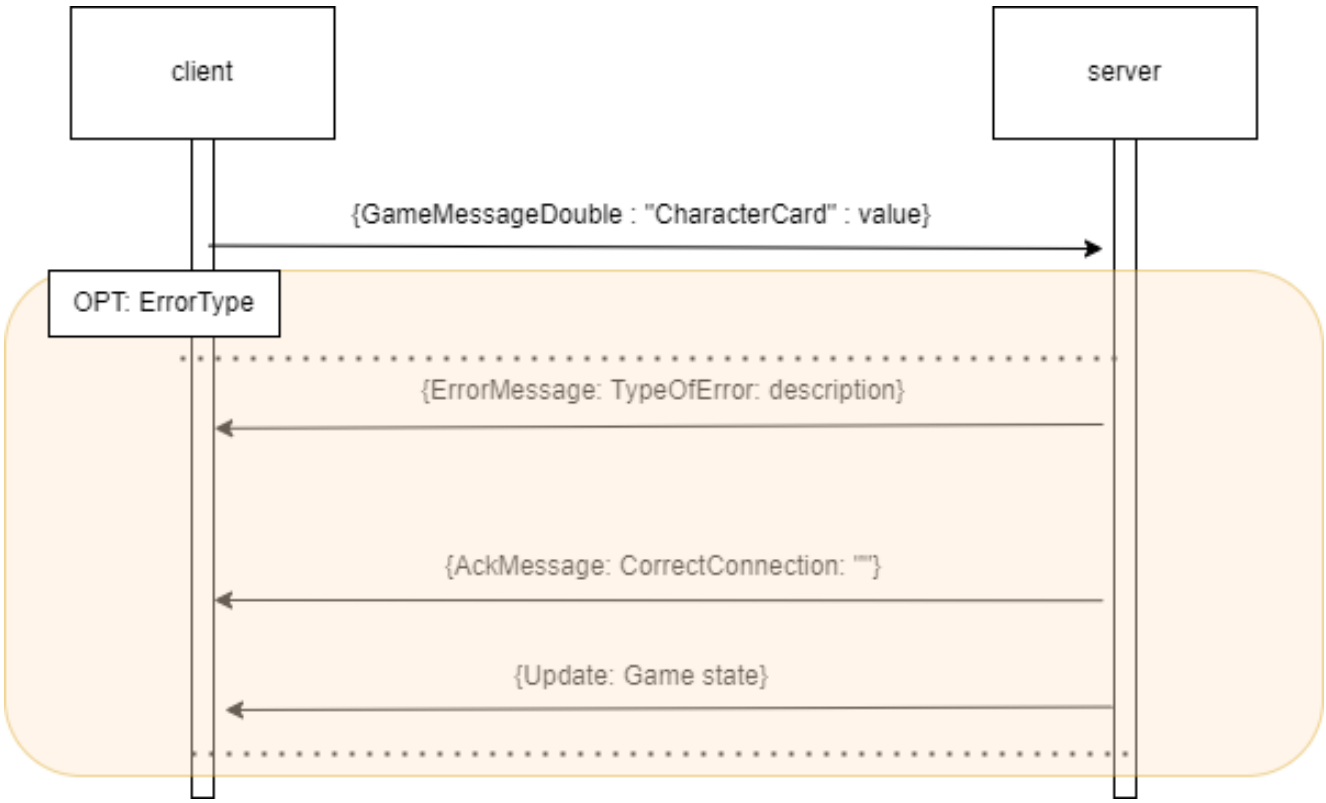


The same happens with PawnMovementMessage:

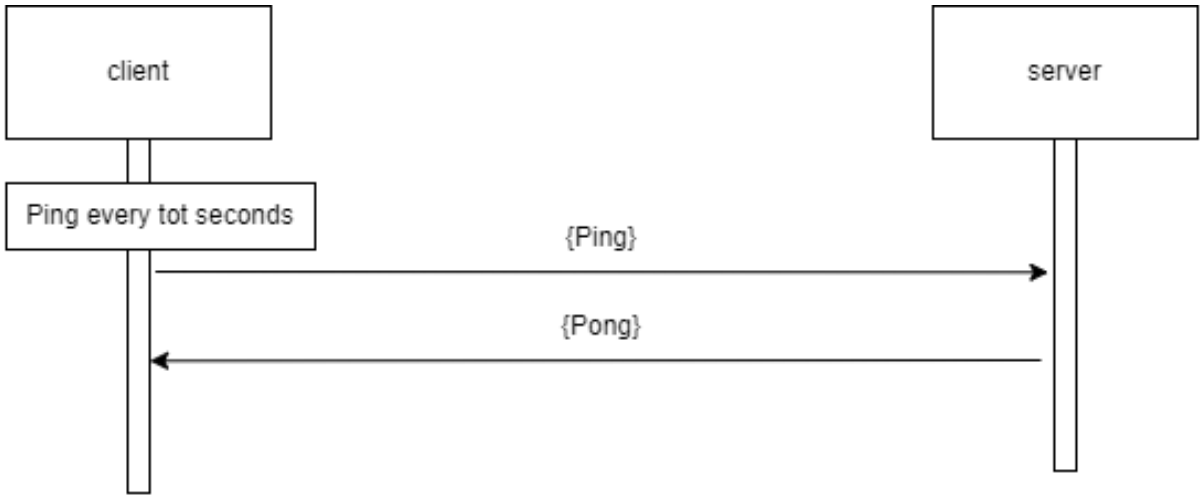


The following sequence diagram represents how the character card are chosen by the client.

The difference with the other game-action choice is that a Character Card can be chosen in every moment of the ActionPhase.



Ping & Pong



AsyncMessage

An AsyncMessage is sent from a client when he wants to leave the game.

There is also the opposite case in which the server sends the AsyncMessage to the client. This happens to inform all the clients about the disconnection of one player or when the server has disconnected.

Disconnection can happen in an explicit way (the one represented on the following sequence diagram) or in other ways (eg. the socket connection is lost, the client exceeds the timeout when he has to make a fame choice, the ping find the lost connection).