Communication protocol - Eryantis

Group PSP15 - Alessandro Ferri*, Alessandro Gabbini**, Angelo G. Gaillet***

Aprile 2022

*alessandro5.ferri@mail.polimi.it; **alessandro.gabbini@mail.polimi.it; ***angelogiovanni.gaillet@mail.polimi.it

1 General description

The communication protocol we implemented is based on Java objects' serialization and de-serialization. In this report, these objects will be called 'messages' as this was the name we chose for them in our project. Each message contains its type (defined by a specific enumeration), and when necessary a payload containing all data that needs to be transmitted. The details of each message are discussed in section 2.

Our network implementation follows an imperative style for the client, meaning that the player is prompted to do each action (except for connecting for obvious reasons) and the client replies with a proper message only when a message initiating a move is received and elaborated. In this model the client does not send messages to the server of its own volition.

Before being elaborated by the controller on the server side the messages are checked for proper formatting and compatibility with the current game phase. This is done in order to better protect against possible malicious clients. This part of the network is not thoroughly discussed in this report as it is out of its scope.

In section 3 of this report the sequence diagrams describing the exchanges of messages during major game phases are provided.

2 Messages

This section contains a list of all implemented messages with a description of their content and when necessary the specifics of its use. Even if not indicated in table 1, messages sent by the client also contain the nickname of the player of said client.

Type	Payload	Source	Additional notes
PING	-	Client	Used to check connection aliveness
DISCONNECTED	-	*	*Generated by the client socket and elaborated by the client when a disconnection event is detected
S_GAMESTATE	Map<String, Map<Color, Integer>> students in entrances Map<String, Map<Color, Integer>> students in dining rooms Map<Integer, Map<Color, Integer>> students on islands Map<Integer, Integer> # of towers per island Map<Integer, TowerColor> color of tower per island Map<Integer, Integer> # of forbidden tokens per island Map<Integer, Map<Color, Integer>> students per cloud Map<Color, String> professor owners int position of mother nature	Server	Generated by the notifyObservers() method in the Game model
S_WIN	String : winner	Server	
S_LOBBY	List<String> : current players int : selected # of players	Server	
S_INVALID	String : error message	Server	
S_ASSISTANT	List<AssistantCard> : available assistant cards	Server	
S_TRYAGAIN	-	Server	Used to signal un unsuccessful action
S_PLAYER	String : nickname of current player	Server	
S_CHARACTER	List<Characters> : available character cards	Server	
S_MOVE	-	Server	
S_MOTHERNATURE	-	Server	
S_CLOUD	-	Server	
S_NICKNAME	-	Server	
R_ASSISTANT	AssistantCard : played assistant String : player's nickname	Client	
R_CHARACTER	Characters : played character String : player's nickname	Client	
R_MOVE	Color : selected student int : selected destination id	Client	
R_MOTHERNATURE	int : destination island id String : player's nickname	Client	
R_CLOUD	int : chosen cloud id String : player's nickname	Client	
R_DISCONNECT	-	Client	Sent at the end of a game when the client wishes to disconnect
R_GAMESETTINGS	int : desired # of players boolean : is this an expert game String : player's nickname	Client	
S_TRYAGAIN	-	Server	
S_PLAYER	String : current player	Server	
S_MONKPRINCESS	CharacterEnum : character name EnumMap<Color, Integer> students	Server	
R_MONKPRINCESS	Color : chosen color CharacterEnum : character name int : island index	Client	
S_JESTERBARD	EnumMap<Color, Integer> : origin EnumMap<Color, Integer> entrance int max movable students CharacterEnum : character name	Server	
R_JESTERBARD	ArrayList<Color> : students from origin ArrayList<Color> students from entrance CharacterEnum : character name	Client	
S_GRANDMAHERBHERALD	CharacterEnum : character name	Server	
R_GRANDMAHERBHERALD	int : island index CharacterEnum : character name	Client	
S_ROGUEMUSHROOMPICKER	CharacterEnum : character name	Server	
R_ROGUEMUSHROOMPICKER	Color : chosen color CharacterEnum : character name	Client	

Table 1: Messages and content

3 Sequence Diagrams

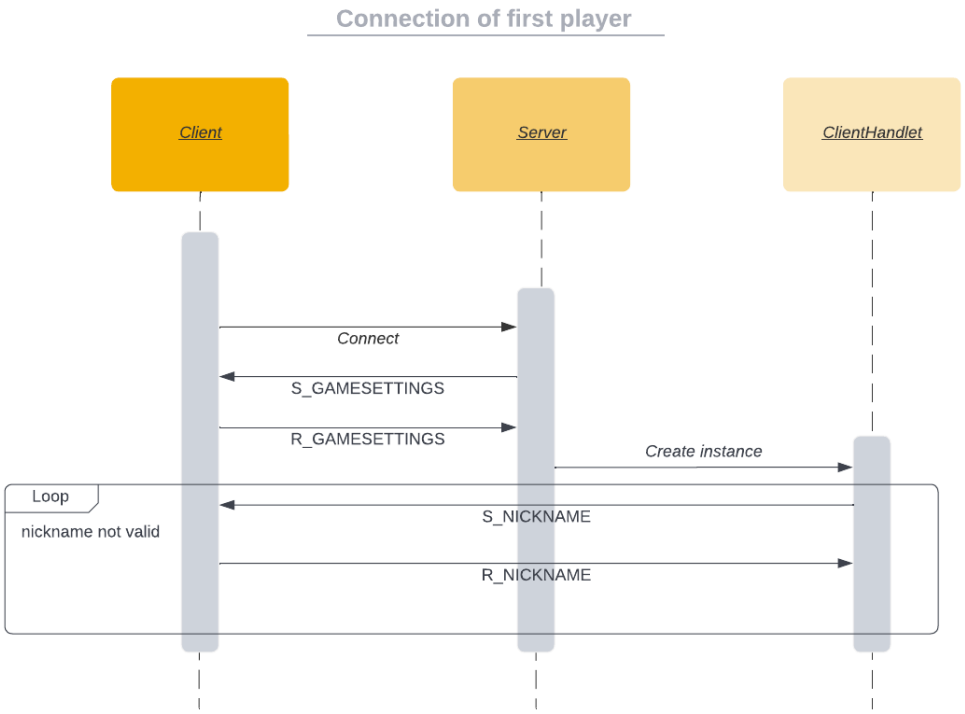

Figure 1: Sequence diagram for the connection of the fist player

For the connection of other players the sequence is the same except for the S_GAMESETTINGS message and the reply message R_GAMESETTINGS which are not sent.

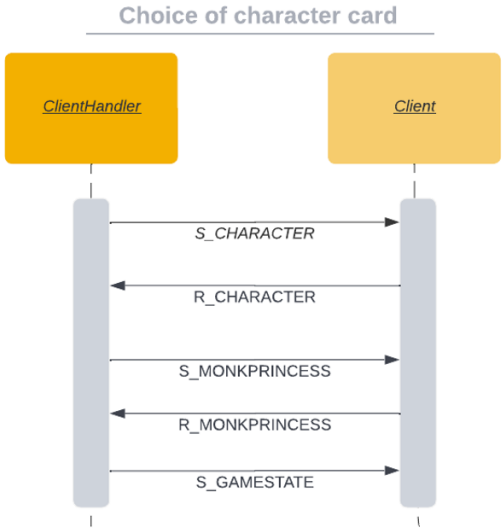

Figure 2: Sequence diagram for character choice

The diagram in figure 2 represents the flow of messages pertaining to the choice of a character card which needs additional parameters to enact its effect. This diagram specifically uses the Monk or Princess character cards as an example. Simple cards which don't need additional parameters only require the S_CHARACTER and R_CHARACTER messages to be exchanged.

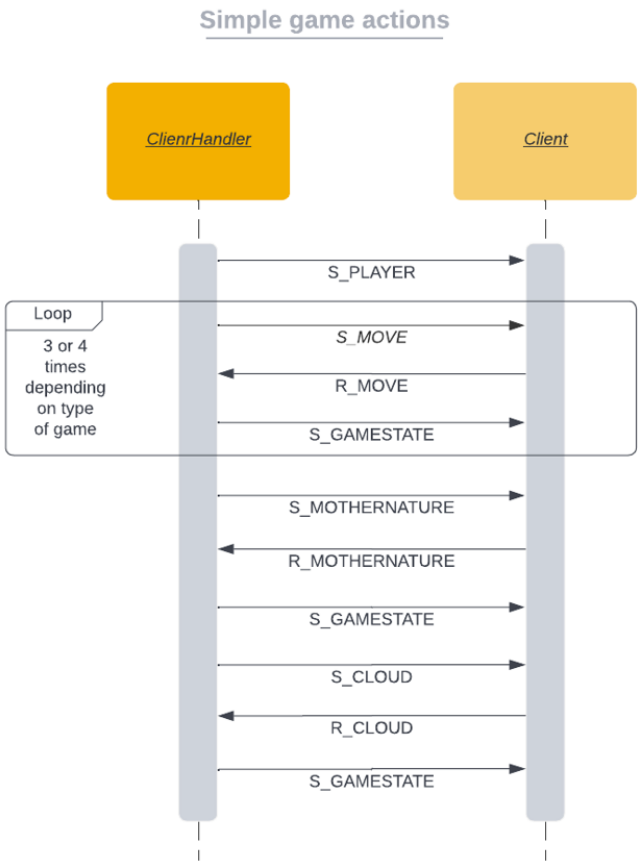

Figure 3: Sequence diagram for simple game mechanics

The diagram in figure 3 shows the client-server interaction for the most common actions performed during a game. The interactions due to character choices are not represented.

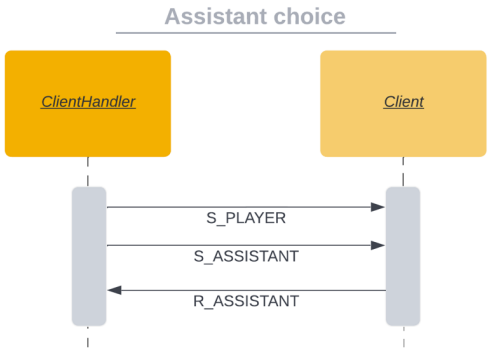

Figure 4: Sequence diagram for assistant card choice

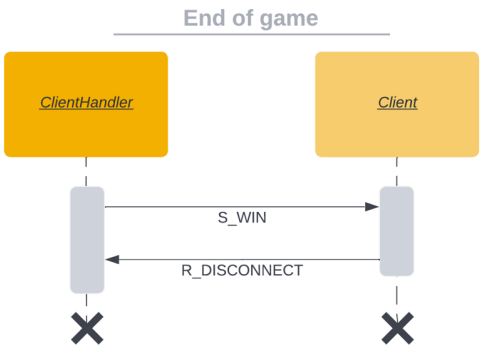

Figure 5: Sequence diagram for end of game