Network protocol AM45

Daniele Dente, Daniele Di Santi, Valerio Donno  
**[implementing multiple games AF]**

Overview

The communication protocol is based on the interaction between a Client and a Server who send messages through the network in a JSON file format. A client always initiates a request, and the server only sends messages if they answer to a previously received message.

Thus, the messages sent from the server could be sent to a user who didn’t previously send a request if the message is of interest to them. The requirement is that the exchange is always initiated by a user.

Example: (This interaction is also shown in the diagrams) A user joins a lobby that previously contained another user. The server answers to the request of the incoming user, and also sends an update to the user already in the lobby.

Every message sent from the Client will contain an idUser and a progressive idRequest. The first is used by the server to determine if the client connected to the socket is the one sending the message; the second is used by the client to check what message the server replied to. Because of that, the server will have to return the idRequest in the reply.

The JSON file format will not be fixed in size, but will vary with each message, containing only the relevant information.

The server reply consists of a copy of the request, in which are appended server specific fields containing the result of the operation.  
This way, the client will know exactly what action had been performed, and the result of said action on the server’s side. This behavior allows for a possibly different implementation of the server, in which rather than denying the action in case of incorrect inputs, it is transformed in the closest approximation of the desired command, and the result of the operation is then sent to the client, which will know what action has been *actually* performed.

The interactions are described in a separate pdf file

Connection states

The main component in the exchange between client and server are the network commands, representing an action. These are grouped into several **connection states.**

At any point, the server and the client can only send and receive a certain subset of messages, depending on what **connection state** they’re in.

Example: While in the lobby, a user can only send the messages READY\_TO\_START, NOT\_READY, START\_GAME, LEAVE\_LOBBY, QUIT

The connection states are the following, allowing the messages contained in brackets.

The command QUIT is available in every state.

* Authentication : [CONNECTION\_REQUEST]
* LookingForLobby : [PLAY\_GAME]
* InLobby : [READY\_TO\_START, NOT\_READY, START\_GAME, LEAVE\_LOBBY]
* StartingGame : [SELECT\_WIZARD, SELECT\_TOWER\_COLOR]
* WaitingForControl : [ ]
* PlanningPhaseTurn : [CHOOSE\_ASSISTANT]
* ActionPhaseTurn : [SELECT\_CHARACTER\*]
  + StudentChoosing : [SELECT\_STUDENT]
  + StudentMoving\*\* : [PUT\_IN\_HALL, PUT\_IN\_ISLAND, DESELECT\_STUDENT]
  + MNMoving : [MOVE\_MN]
  + CloudChoosing : [CHOOSE\_CLOUD]
  + EndTurn : [END\_TURN]
* CharacterCardActivation : [SELECT\_ENTRANCE\_STUDENTS, SELECT\_STUDENT\_COLORS, SELECT\_ISLAND\_GROUP, SELECT\_STUDENTS\_ON\_CARD, PLAY\_CHARACTER]

\* Action only available in an advanced game  
\*\* Character card selection disabled while moving a student

Network commands index

* Connection management
  + **Quit**
  + **ConnectionRequest [ACK: AcceptConnection] [ERR: ConnectionDenied]**
* **Lobby Commands**
  + **PlayGame [ACK: AddedToLobby]**
  + **ReadyToStart [ACK: GameIsStarting] [ERR: GameNotStarting]**
  + **NotReady**
  + **StartGame [ACK: GameStarted] [ERR: PlayersNotReady]**
  + **LeaveLobby**
* **Game Initialization Commands**
  + **SelectWizard**
  + **SelectTowerColor**
* **Game Commands**
  + **ChooseAssistant**
  + **SelectStudent**
  + **PutInHall**
  + **PutInIsland**
  + **DeselectStudent**
  + **MoveMotherNature**
  + **ChooseCloud**
  + **EndTurn**
  + **SelectCharacter [ERR: NotEnoughCoins]**
  + **SelectStudentColors**
  + **SelectEntranceStudents**
  + **SelectIslandGroup**
  + **SelectStudentsOnCard**
  + **PlayCharacter**
* **Server-only commands**
  + **ServerLobbyStatus**
  + **ServerLobbyStart**
  + **ServerGameInitializationStatus**
  + **ServerGameStart**
  + **ServerYourTurn**
  + **ServerGameUpdate**
  + **ServerUserDisconnected**
  + **ServerGameWon**

Command description

Follows a list of the commands above, each with their required network fields in curly brackets.  
If some parameters are mentioned in the section introduction, then those parameters are required for all the commands in that section.

Connection Management { idRequest : int }

* Quit {idUser : int} : The user closes the program and sends a notification to the server
* ConnectionRequest { nickname : String } : A connection is requested from a new user

Lobby Commands { idUser : int , idRequest : int }

* PlayGame { rules : GameRuleEnum } : The user requests to play a game with the specified rules
* ReadyToStart { } : The user sends a message notifying the server they are ready to start the current game
* NotReady { } : The user sends a message notifying the server they are no longer ready to start the current game
* StartGame { } : The game host requests the game to start. If a non-host sends this command, they will receive a negative answer
* LeaveLobby { } : The user requests to leave the current lobby

Game Initialization Commands { idUser : int , idRequest : int }

* SelectWizard { idWizard : int } : The user requests to select a wizard (if available)
* SelectTowerColor { idTowerColor : int } : The user requests to select with which color tower (and consequently in which team) they want to play

Game Commands { idUser : int , idRequest : int }

* ChooseAssistant { idAssistant : int } : The user requests to play an assistant during the planning phase
* SelectStudent { position : int } : The user selects a student from their entrance during the action phase
* PutInHall { } : The user requests to move the (previously) selected student to their dining hall
* PutInIsland { idIslandGroup : int } : The user requests to move the (previously) selected student to the selected island group
* DeselectStudent { } : The users requests deselect a previously selected student
* MoveMotherNature { steps : int } : The users requests to move mother nature by the specified number of steps
* ChooseCloud { cloud : int } : The user chooses from which cloud they will take the students to refill their entrance.
* SelectCharacter { characterCardPos : int } : The user asks the server to select the character card and send the requirements for that card
* SelectStudentColors { studentColors : StudentEnum[] } : The user selects student colors (as a requirement for a character card activation)
* SelectEntranceStudent { atEntranceSelected : int[] } : The user selects the student at the given positions of their entrance (as a requirement for a character card activation)
* SelectIslandGroup { idIslandGroup : int } : The user selects an island group (as a requirement for a character card activation)
* SelectStudentOnCard { onCardSelected : int[] } : The user selects the student at the given position on the selected character card (as a requirement for a character card activation)
* PlayCharacter { characterCardPos : int } : The user confirms their selection (if any were made) and requests the activation of the character card

Server commands { asyncIdRequest : int}

* ServerLobbyStatus { beanType : BeanEnum, bean : Bean } : The server sends the updated lobby status
* ServerLobbyStart { } : The server notifies the user that the host started the game
* ServerGameInitializationStatus { beanType : Class, bean : Bean } : The server sends the updated game initialization status
* ServerGameStart { } : The server notifies the user that the proper game started
* ServerYourTurn { asyncGamePhase : PhaseEnum } : The server notifies the user that it’s their turn to play the phase sent with the message
* ServerGameUpdate { asyncView : VirtualViewBean } : The server sends the updated game status
* ServerUserDisconnected { asyncIdUser : int, asyncUserNickname : String } : The server notifies the user that a client disconnected, along with its id and nickname
* ServerGameWon { asyncWinner : TeamEnum } : The server notifies the user that a team won the game.

Ping routine

To enable solid communication between the two parties, the user and the server will periodically send ping messages to notify each other of their aliveness.

The timeout is set for both parties at 5 seconds, and if this time elapses without a reply, then the connection is interrupted.  
The time between pings is set at 1 second.

Unimplemented features

The following communication protocol features have not been implemented due to time constraints, but the structure of the program allows for an easy implementation, in case of further development:

* To ensure the stability of the protocol, the Client will always have a set timeout for each request, to guarantee that a request gets resent if no answer is provided by the Server within a certain time. This may happen when the Server receives too many requests, and its buffer becomes full before receiving the Client’s message.
* Unfortunately, as we’ve had to undergo a complete overhaul of the network protocol, remnants of the old protocol are still left in the code, which would need to be cleaned