Communication Design documentation

Group AM 38:
Valeria Amato, Nicolò Caruso, Francesco Barisani

Index of contents:

Brief note on implementation approach

In the following document, we explain our design for the communication aspects of our game.
To allow for a better understanding, we preface that we decided to follow the JSON-driven approach for the implementation of the messages.
Therefore, we designed messages with a specific pattern of fields and allowed values. The messages will be transformed into json files and then sent through the network.

Message types list

We designed a message categorization to sort all the messages by their type, semantics and occurrence within the phases of the game.
Here are the main types of messages we handle.

OK - positive generic acknoledgment answer
NACK - negative generic answer
ERROR - negative answer with subtype and explaination (e.g. if the action is not allowed)
PING , PONG - check connection stability (ping request to the client and response to the server)
CONNECTION - messages for connection and “network” operations between the sockets
ACTION - actions of the game that the player can use during its turn
SET - set some parameters in the initial phase of the game
END_MESSAGE - states that the game is ended, providing information on the ending status
UPDATE - notifies the view of the clients updating it after a model state change
REQUEST
ANSWER

Messages details and categorization (reference table)

MessageTypeEnum (int)	**MessageSubtype**	**Payload (content)**	**NOTES**
OK (0)	—	—--	
NACK (1)	__	—	
ERROR (2)	[ErrorType]		
	GenericError	String errorString	
	ConnectionError	String errorString	
	InvalidInput	String errorString int minVal ; int maxVal	
	FatalError	String errorString	
	NicknameAlreadyTaken		
	LobbyIsFull		
	NumberOfPlayersAlreadySet	String errorString int actualNumOfPlayers	
	GameModeAlreadySet		ErrorString contains “ActualGameMode”
	AlreadyUsedAssistantCard	String errorString	ErrorString = {“AlreadyUsedByOtherPlayer” ; “AlreadyDrawnBySamePlayer” }
	AllMovesUsed	String errorString	
	NoSuchStudentInSchoolEntrance	String errorString	ErrorString can say the students present in the schoolEntrance
	DiningRoomColorFull	String errorString	
	IslandOutOfBound	String errorString	
	ExceededStepsNum	String errorString	
	CardAlreadyUsed		

	NotEnoughCoins		
	NotValidOrigin		“origin” refers to the “place” from which the move is taking students
	NotValidDestination		
	IslandAlreadyBlocked		
	AlreadyUsedSpecialCard		only one each turn
PING (3)			Sent to all the clients to check whether they are connected
PONG (4)			Replied by each client when a PONG is received
CONNECTION (5)	[ConnectionTypeEnum]		
	RequestConnection		
	AcceptConnection		
	CloseConnection		
ACTION (6)	[ActionTypeEnum]		
	UseAssistantCard	int priority	Each card is unambiguously identified by its priority.
	UseSpecialCard	String specialCardName	
	MoveStudent	Color color StudentCounter from StudentCounter to, int position	
	MoveMotherNature	int islandPosition	
	ChooseCloud	int cloudPosition	
	ChooseTilePosition	int islandPosition	choose the island where to place the NoEntryTile
	ChooseColor	Color color	
SET (7)	NickNameMessage	String nickname	
	SelectNumberOfPlayers	int numOfPlayers	
	SetGameMode	boolean isExpertMode	

END_OF_GAME_MESSAGE (8)			
	–	String endOfGameString	
UPDATE (9)	[UpdateTypeEnum]		
	SetupUpdate	Map <String, Tower> playersMapping int numPlayers boolaen isExpertMode	create an association between the nickname and the specific SchoolBoard
	CloudViewUpdate	int position Map <Color, Integer> studentsMap int limit	
	IslandViewUpdate	int position Map <Color, Integer> studentsMap Tower towerColor int numTower boolean isDisabled	
	ArchipelagoViewUpdate	int numIslands int motherNaturePosition	
	SchoolBoardUpdate	Map <Color, Integer> diningRoomMap Map <Color, Integer> schoolEntranceMap Tower towerColor int numTower int coins	towerColor uniquely identifies the player
	ProfessorsUpdate	Map <Color, Tower> professors	
	SpecialCardUpdate	int coinCost SpecialCardName name Map <Color, Integer> studentsMap int numTiles	
	PhaseUpdate	phaseEnumValue phase	
	AssistantsCardUpdate	Tower playerTower ArrayList <Integer> availableCards	playerTower identifies the player whose deck has been updated
	LastUsedAssistantCardUpdate	Tower playerTower Integer lastUsed	playerTower identifies the player
	CurrentPlayerUpdate	String nickname	
	LeaderboardUpdate	boolean isEndOfGame Map <String, int> leaderboard	Notifies the current leaderboard status (also at the end of

			match)
REQUEST			
	LobbyRequest		
	GameModeRequest		
	PlayersNumberRequest		
ANSWER			
	LobbyAnswer	int numOfPlayers	MessageGenerator. numberPlayerlobby Message
	GameModeAnswer	boolean isExpert	
	PlayersNumberAnswer	int numOfPlayers	

Explanation and notes on the message table

- Every message is composed of 3 major fields :
    - MessageTypeEnum
    - MessageSubtype (it is optional)
    - Payload : contains all the values and parameters carried by the message

- MessageTypeEnum is an enum that lists all the main message types (see "Message types list"). It is the first field of the message and is used to "filter messages" and process the message according to its type.
  Therefore, the fields expected in the payload can be predicted by checking the MessageType field.

- Every MessageSubtype extends a specific MessageType.
  Some MessageTypes, such as "OK", do not have a MessageSubtype.

- In the JSON string we transmit, both MessageType and MessageSubtype are represented by an integer obtained by calling the method valueOf() on the enum. As an example of that, we associate the integer value to its corresponding MessageType in the first column of the table above.

- ERROR type messages contain many subtypes of errors, each dealing with a specific kind of error, some of which derive from exceptions that occur within the model. Broadly speaking, these error messages are triggered by any scenario in which the Server replies to an invalid request or action attempted by the client, ranging from "technical" issues - such as connection issues - to internal-logic issues pertaining to the rules of the game, as implemented in the Model.

- UPDATE type messages are the messages that notify the Clients that some change has occurred within the status of the Model.
  Those updates also include the new status that the view of the clients should render, according to the status of the Model.

  The updates range from changes in the status of tiles and objects of the game, to changes in the game phases and rounds, such as the current player or the current phase of the game.

- When the json message is created, the Enum fields in the message do not contain the enum value per se, but the corresponding “odinal” integer, obtained using the ordinal() method

---

Example of messages format and pattern

Here are some examples of how the message patterns discussed above should look like and how they are represented when trasmitted as a json file.

A generic error message should look like as follows:

GenericErrorMessage (int messageType, int errorType, String errorMessage)
{ “messageType”: 2, “errorType”:0, “errorMessage”:”ERROR- Generic error message string”}

A generic OK message is as follows :

OKMessage (int messageType)
{“messageType”: 0}

An update message that notifies a change in the third cloud (position = 2) that results in a new distribution of students in the cloud should look as follows :

CloudViewUpdate (int messageType, int updateType, int position, Map <Color, Integer> students, int limit)
{ “messageType”:9, “updateType”:1, “position”:2,
“students”:{"BLUE":2,"YELLOW":0,"PINK":0,"RED":1,”GREEN”:0}, “limit”:3 }

Game phases

In the design of the communication, we considered the game phases in the following list. Each phase refers to a specific interaction between the server and the clients.
To further detail the interaction dynamics within said phases, we have designed the Sequence Diagrams of the following phases: “Login phase”, “Planning phase” and “Action Phase”

Game Phases list :

1. Login phase
2. Setup phase and initialisation
3. “Planning phase”
4. “Action phase”
    a. 3 students moves
    b. move MotherNature
    c. Choose a new Cloud
    d. EndOfTheGame (different end scenarios)
5. Ping - Pong : repeated every X seconds to check connection status
6. Special card - asynchronous use of SpecialCard (can take place within the Action Phase)

Advanced Functionalities interactions :

7. Multi-games
8. A.F. “Persistence” : save the Game status, wait for reconnections (check the nicknames!) , then restore the Game status

---

Sequence Diagrams

Sequence Diagrams with detailed interaction can be found as separated files.
Note that the messages shown in the sequence diagrams follow the categorization presented above.