Communication Protocol

Introducion

This document describes the Communication Protocol between Client and Server used in our implementation of the Adrenaline Game. More information about this project is avaiable at https://github.com/GetnaG/ing-sw-2019-abbo-attolini-bentej.

The communication protocol follows the following phases:

- Login Phase
- Join Game Phase
- Turn Phase

This protocol has been designed following what we call “The Ask To Choose / Choice” paradigm. Once the client is connected, the server always takes the initiative in the conversation. The client has to choose from a finite set of provided answers, except some rare cases such as Username input.

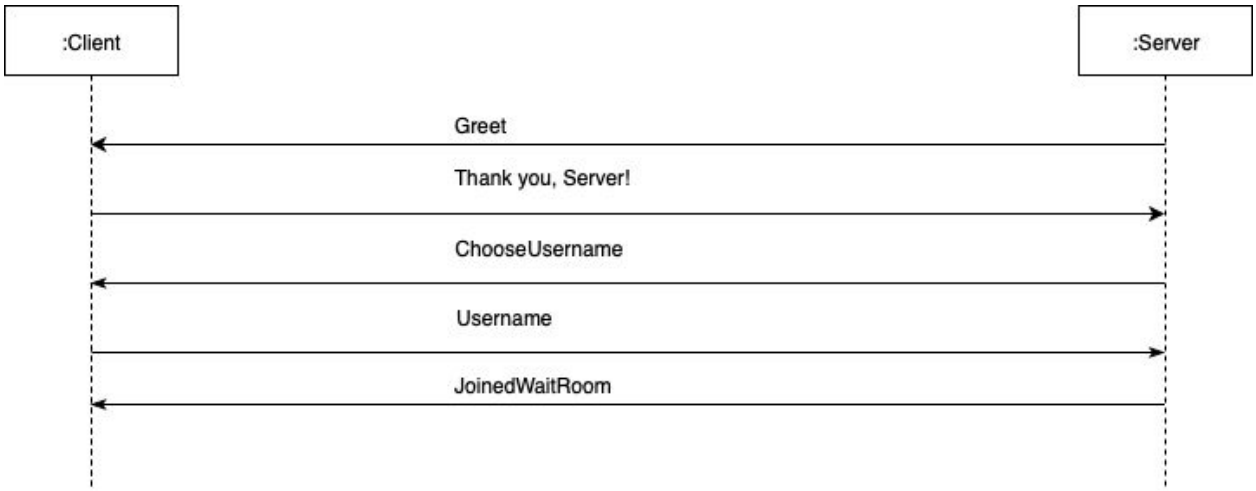
The communication protocol is “view-agnostic”. Therefore, it will work on both CLI and GUI. However, some messages may contain some information that is not used by CLI. This means that it’s not the Protocol job to filter the information according to the View used.

The protocol uses JSON to define the structure of its messages. Every message contains a command field which specifies the type of the message. Thus, by looking at this field, any party can infer the structure of the message.
The most important commands are `notify` and `update`. The former takes care of notifying that an event happened, but it hasn’t affected the state of the game. For instance, a player has disconnected. The former deals with the change in the state in the game. The reader can learn more about these two commands in the following pages.

Phases

Login Phase

The client connects to the server using Socket or RMI. Once the server receives the connection, it greets the Client like in a handshake.



Message	**Content**

Command:Greet	{
	command: GREET,
	hasMessage: true,
	Message: "Hello Client! You are now connected."
	}

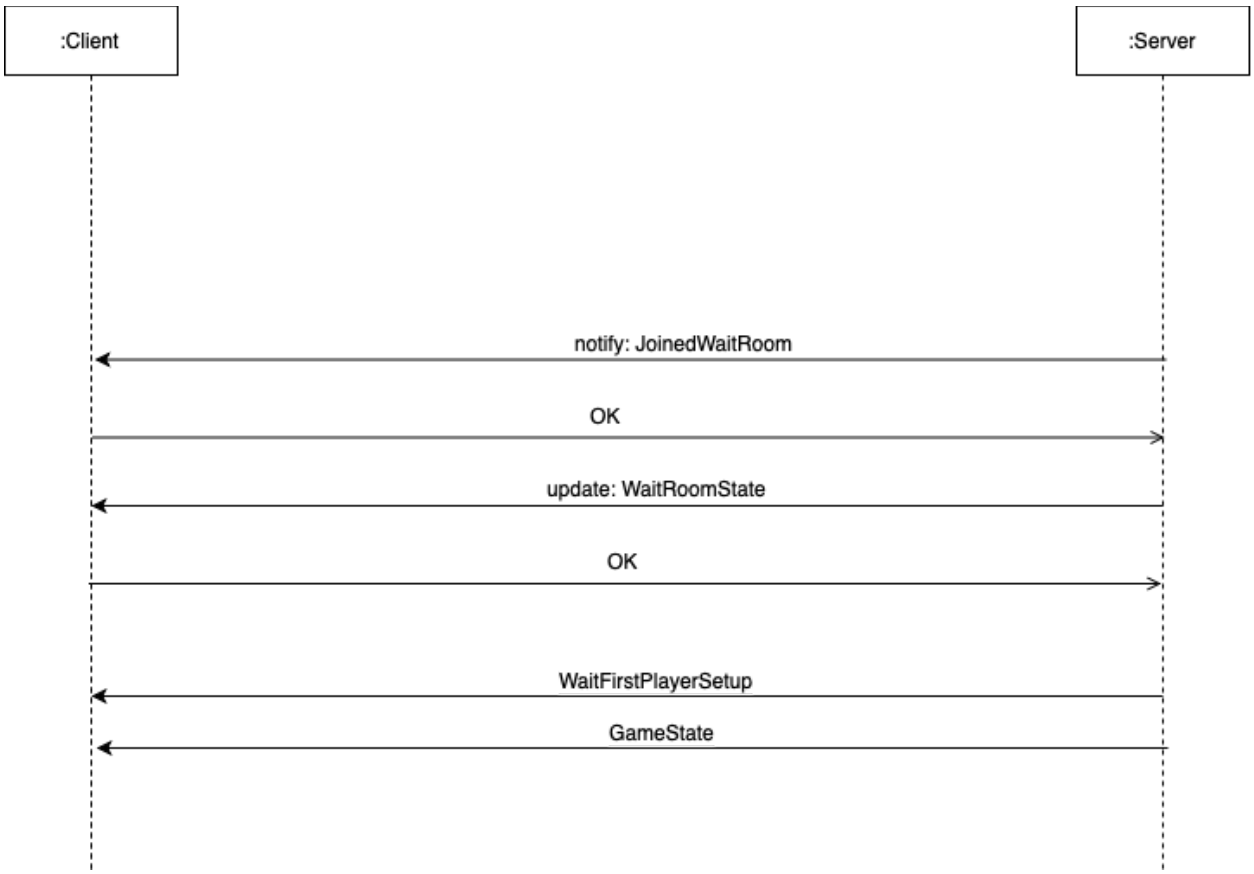
Command:Thank you, Server.	{
	command: GREET,
	hasMessage: true,
	Message: "Thank you, Sever."
	}

AskToChoose:Username	{
	command: USERNAME,
	hasMessage: true,
	Message: "Choose a username"
	}

Choice:Username	{
	command: USERNAME,
	hasMessage: true,
	Message: "Bob"
	}

Join Game Phase

Once the client joins to the room, the server keeps sending update about the waiting room state.



Message	Content

Notify: JoinedWaitRoom	`{ command: NOTIFY`
	`notifications:`
	`[`
	`{`
	`type: JoinedWaitRoom}`
	`]`
	`}`

Update: WaitRoomState	`{ command: UPDATE`
	`updates:`
	`[`
	`{`
	`type: WaitRoomState,`
	`State :value`
	`}`
	`]`

	}

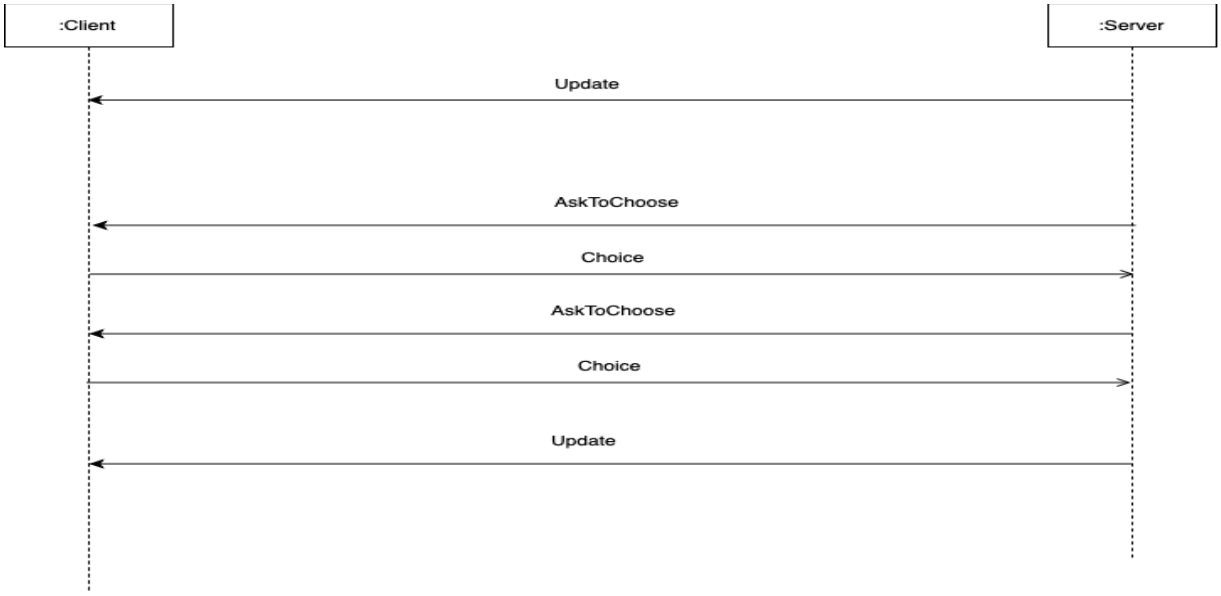
Notify: WaitFirstPlayer	{ command: NOTIFY
	notifications:
	[
	{
	type: WaitFirstPlayer}
	]
	}

Update: GameState	{ command: UPDATE
	updates:
	[
	{
	type: GameState,[attr1:value1,...]
	}
	]
	}

Turn Phase

In this phase there is an extensive use of the "AskToChoose/Choice" paradigm. The player is invited to choose from a finite set of possible answers.

The following diagram describes the case in which it's the turn of the player.



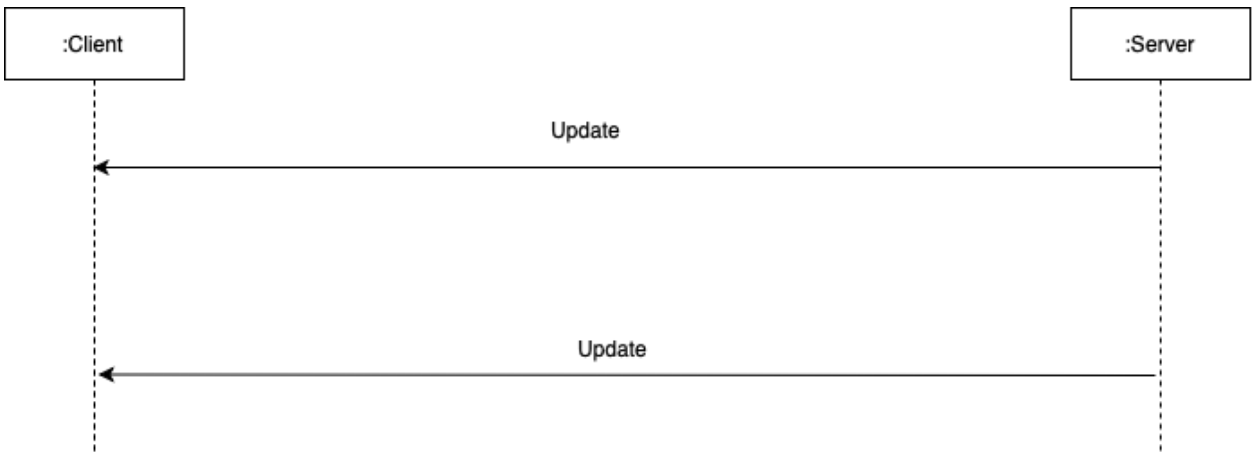
Message	Content
Update: GameState	{ command: UPDATE updates:

	[
	{
	type: GameState,[attr1:value1,...]
	}
	]
	}

AskToChoose	{
	command: COMMAND_NAME,
	hasOptions: true,
	options: [
	["opt1 1", "opt1 2"],
	["opt2"],
	]
	}

Choice	{
	command: COMMAND_NAME,
	hasOptions: true,
	options: [
	["opt1 1", "opt1 2"]
	]
	}

The following diagram describes the case in which it is not the turn of the player.



Message	Content

Update: GameState	{ command: UPDATE
	updates:
	[
	{
	type: GameState,[attr1:value1,...]
	}
	]
	}