Network Protocol for Uno

Mentors:

Anton Tsankov
Valeri Andreev
Maarten Looijenga

Maintainers:

Mehdi Hajidehabadi (s2656930)
Andrey Nikolov (s2955369)

Discussion Leaders:

Umair Aamir Mirza (s2928558)
Felix van Delden (s3087573)

Documentation and Review:

Tom Hansult (s2993074)
Arthur da Rosa-Peters (s2914093)
Yousef Kaid (s2601257)

Introduction:

This document contains the necessary specifications regarding the network protocol for Uno as discussed during the Protocol Session that took place on 01/12/2023. The ideas that were discussed and presented during that protocol session were taken into account during the development of this network protocol. As all ideas cannot be directly implemented, the protocol maintainers tried to make the protocol as general yet functional as possible, to accommodate the dispersion in the ideas presented. This document is to be accompanied by the relevant Java files that contain the implementation of the protocol messages and methods, with detailed JavaDoc comments to explain how exactly to implement the protocol in the most efficient manner.

The report begins by providing a general insight into what the major ideas were doing into the development of the protocol, followed by the necessary protocol messages (also accounting for data types within the game of Uno), an analysis of the exceptions (and handling techniques), additional notes that may be helpful, along with a relevant flow diagram to illustrate the processes. The report will be concluded with a short discussion regarding design choices, feature completeness, and any patterns that are supported.

Discussion of Relevant Development Ideas:

Before any of the protocol was developed, a discussion was required around the operations of the server and the client, to determine the preliminary exchanges that occur before the game can be developed. Once a basic understanding of these correspondences was reached, it was extended to include the optional features in the project, and was later improvised to make processes as detailed yet efficient as possible. Once a standard for message codes was agreed upon, the respective tables were made to accommodate correspondences between client and server, and a discussion regarding the data in Uno led to an implementation of specific data formats to be used within the messages being transmitted. All in all, the protocol attempts to facilitate a stable, functional, and robust implementation of Uno over the network.

Some of the questions asked during the course of development were:

1. What information does the client require?
2. What data does the client possess?
3. What information does the server require?
4. What data does the server possess?
5. What are the fundamental exchanges required for forming a connection between an Uno Client and Uno Server?
6. What data within the game must be accounted for during the transmission of data?
7. Is the data being transmitted as Strings or Objects?
    a. Why was this particular type selected?
8. What additional features can be facilitated through this protocol? If so, what are the ways in which this can be done most effectively?

Accounting for Data in Uno:

In order for the protocol to be successful, it requires a strict format to abide by. The chosen data medium to send correspondences was String, as agreed upon during the protocol session. As this is the case, Strings are very particular regarding character positioning and deconstruction: the appropriate character format must be used to ensure that the client and server are not running into issues regarding data processing, which can be seen in the Java code provided. That being said, the following table contains relevant information about the Uno data presentation, which will be useful later on.

Name	Format	Description	Example
Color	[COLOR]	[COLOR] must be replaced with the corresponding color in uppercase letters.	YELLOW
Card	[COLOR] [VALUE]	[COLOR] must be replaced with the corresponding color of the card in uppercase letters. [VALUE] must be replaced with the corresponding value of the card in uppercase letters. The value can be the number on the card in digit or a special card name. **Note**: There has to be a space character(“ ”) between the two variables.	RED DRAW_4 YELLOW WILD RED 0 GREEN REVERSE
Player name	[player_name]	[player_name] must be replaced with the player's unique name in lower case.	david
Player info	[player_name]:[number_of_cards]:[score]	[player_name] must be replaced with the player's unique name. [number_of_cards] must be replaced with the number of the cards in the player's hand. [score] must be replaced with the player’s score. Note: There has to be a column character(“:”) between the different variables.	andrey:5:75 robert:7:55 david:3:125
Argument Delimiter	|	The delimiter(“|”, vertical bar) indicates the separation of arguments. **Note**: In the protocol tables below there are examples that include “ | ” with space characters before and after, which is for only the sake of explanation, better look and reading of the examples. Therefore, you should not include the space characters during the data transmission.	MH|david|human_player
List Delimiter	;	This character(“;”) in an argument means it is a list. Therefore, the argument should be converted from a list to a string before sending the protocol and vice versa after receiving it, following the Java best practices.	RED 1;BLUE SKIP
List sub-item delimiter	:	This character(“:”) in a list separates the different attributes of that list. Therefore, the variables should be concatenated in the correct order to a string joined by “:” before sending the protocol and split into variables after receiving it, following the Java best practices.	andrey:5:75;robert:7:55;david:3:125
Your turn	[is_your_turn]	[is_your_turn] is replaced with “true” or “false”.	false
Game mode	[game_mode]	[game_mode] is replaced with “normal”, “progressive”, “jump_in” or “seven_o”	seven_o
Strategy	[strategy]	[strategy] is replaced with the name of the different strategies your computer players can have. The value of this argument differs based on the strategies you implement and how you name them.	
Direction	[direction]	[direction] is replaced with “true” or “false” while “true” represents clockwise and “false” counter clockwise direction.	false
Lobby name	[name]	[name] must be replaced with the corresponding name.	bit_students myLobby
Player type	[player_type]	[player_type] must be replaced with “human_player” or “computer_player”.	human_player
Chat message	[message]	[message] must be replaced with the corresponding message.	Nice move

Relevant Design Choices:

In the table above, the reason for keeping a space between Card Color and Value was because it would enable simpler representation of the information on the client-side as opposed to having to deconstruct and again concatenate the String to present it. The protocol does not specify exactly where formatting of information takes place, so this should be done in accordance with best practices where applicable. Additionally, it is important to note that within a list, the semicolon is used to split each particular argument, and then commas can be used to further split the subarguments. This was specified to ensure compatibility across implementations, ensuring that there is one agreed upon mechanism for displaying and transmitting game information and related information.

From **Server** to **Client**:

The discussion of the protocol codes begins with the correspondences from the Server to the Client, as shown in the table below. Relevant hyperlinks are added to allow for convenience regarding data presentation.

Code	Action	Description/Arguments	Example
AH	Accept Handshake	To inform the player that their handshake request was accepted.	AH
IAD	Inform Admin	Admin only: Send a request to the player, that he is the admin of the game - so he can add computer players, start the game, etc.	IAD
BPJ	Broadcast Player Joined	1. Player name (see format)	BPJ | mehdi
GST	Game Started	1. Game mode (see format)	GST | normal GST | jump_in
RST	Round Started		RST
BGI	Broadcast Game Information	1. Top card (see format) 2. Corresponding player's hand (see format card, list) 3. List of players of the game sorted by the order of turn (see format player info, list, list sub-item) 4. Your turn: indicates if it is the player's turn (see format)	BGI | RED WILD | RED 6;BLUE REVERSE | andrey:5:75;robert:7:55;david:3:125 | false BGI | YELLOW 3 | GREEN 6;GREEN SKIP | robert:7:55;david:3:125;andrey:5:75 | true
BCP	Broadcast Card Played	1. Player name (see format) 2. Card (see format)	BCP | david | YELLOW 3
BDC	Broadcast Drew Card	1. Player name (see format)	BDC | david
BTS	Broadcast Turn Skipped	1. Player name (see format)	BTS | david
BRS	Broadcast Reverse	1. Direction (see format)	BRS | true
BLG	Broadcast Left Game	1. Player name (see format)	BLG | david
RP	Remind Play	Indicate that the server is expecting a response from the player whose turn it is. 1. Time left: the time in seconds left for the player to play otherwize their turn is skipped	RP | 30
RE	Round Ended	1. Name of the winner (see format)	RE | andrey
GE	Game Ended	1. Name of the winner (see format)	GE | andrey
ERR	Send Error Code	1. Error code (see table)	ERR | E0001

Relevant Design Choices:

1. Referencing BGI in the table above, the reason for keeping all of these four information pieces in one transmission is to ensure that the view of the client is updated to the latest version when they are playing their turn. This means that if a client does not receive the latest version, then neither receive the permission to request input data, so it prevents the game from crashing. Note that necessary precautions still need to be taken on the server-side to ensure that late responses and other information exchanges are monitored and dealt with. These should be handled with respect to best practices. Furthermore, the reason for keeping the list of players in playing order is to provide some semblance of the game direction, and so that it is easier for the client-side TUI to show without needing to know the index position and direction of the game.
2. BTS either occurs when a player is skipped by a Skip card, or they did not make a decision in time.

From **Client** to **Server:**

The table below focuses on the protocol messages that the client may send to the server, addressing specific details in each case.

Code	Action	Description/Arguments	Example
MH	Make Handshake	1. PlayerName (see format) 2. PlayerType (see format)	MH | david | human_player
ACP	Add Computer Player	Admin only 1. Name of the computer player (see format) 2. Strategy (see format)	ACP | bot_1 | advanced
SG	Start Game	Admin only 1. Game mode (see format)	SG | progressive
PC	Play Card	1. Card (see format)	PC | RED 4 PC | BLUE REVERSE
DC	Draw Card	Draw a card from the deck	DC
LG	Leave Game	Terminate the connection of the player	LG

Relevant Design Choices:

The reason for implementing specific commands for admin players relates to the integration of lobbies, an extra feature which is disclosed below. Separating the player's decision into drawing a card or playing a particular card was done so that it is easier to manage on the server-side. The player that creates the lobby becomes the admin.

Extra Features:

The following table contains protocol messages for the extra features, which can be used to facilitate the integration of extra features into the client-server programs.

From	To	Code	Action	Description/Arguments	Example
Lobby:					
Server	Client	LOL	List Of Lobbies	1. List of lobbies names (see format) -OR- 1. List of lobbies names and the their number of participants (see format name, list, list sub-item)	LOL | my_lobby;bit_students -OR- LOL | my_lobby:5;bit_lobby:12
Client	Server	CL	Create Lobby	Allows players to make a lobby with a specific name. 1. Lobby name (see format)	CL | my_lobby
Server	Client	BCL	Broadcast Created Lobby	1. Lobby name (see format)	BCL | my_lobby
Client	Server	JL	Join Lobby	Allows players to join a lobby. 1. Lobby name (see format)	JL | my_lobby
Server	Client	BJL	Broadcast Player Joined Lobby	1. Player name (see format)	BJL | david
Chat:					
Client	Server	SM	Send Message	Allows all player to send a message in the chat 1. Message (see format)	SM | Hello world
Server	Client	BM	Broadcast Message	Broadcast the message sent from a player to other players. 1. Message (see format)	BM | Hello world
Game:					
Client	Server	UNO	Say 'UNO'	Say UNO is a mandatory activity that the player with 1 card remaining has to do. If he doesn't he is punished by drawing 2 cards.	UNO
Server	Client	BUNO	Broadcast Say 'UNO'	1. Player name (see format)	BUNO | david

Relevant Design Choices:

The table above helps understand how the protocol supports extra features, and it is important to distinguish between these commands, especially regarding the creation of lobbies which needs to be managed well with the server-client correspondences.

Errors:

The following table contains all the necessary error codes that the protocol needs to know (both the client and server need to know these error codes so that they can respond effectively). Within the Java code provided, general exceptions are referenced, however, it would also be relevant to discuss how such error codes need to be handled (which is a column in this table). How these required actions are dealt with is subject to the design of each game individually, and should comply with best practices. Certain exceptions can be created in relation with the protocol error messages in compliance with best practices, and the messages provided in the table can be used along with the error codes to create unique exceptions stemming from the enum of error codes. However, this protocol is not liable for any external exceptions that may occur during the development of the game or any external elements, such as NullPointerExceptions, IllegalArgumentExceptions, and so on.

Code	From	To	Description	Message	Action Required
E001	Server	Client	The input does not follow the protocol format required	"Protocol violated."	Try again
E002	Server	Client	The PlayerName chosen is already in use, the server requires unique player names.	"The player name [PlayerName] has been taken. Please enter another name."	Ask the client to re-enter the Player Name.
E003	Server	Client	The input provided to the server is missing parameters/arguments of the protocol.	"Parameters/arguments are missing or violate protocol. Please re-enter."	Ask the client to enter all necessary commands.
E004	Server	Client	The game has already begun (players cannot join this game).	"The desired game is already in action. Please choose another game."	Tell the client that the game is in progress.
E005	Server	Client	The lobby for the desired game is full (player cannot join this game).	"The desired game is at maximum capacity. Please choose another game."	Tell the client that the lobby is full.
E006	Server	Client	The command is invalid, not following conventions of delimiter, or other.	"Invalid input. Please type a valid input."	Tell the client that input is invalid.
E007	Server	Client	A player has disconnected from the server (unresponsive).	"Player [Name] has disconnected"	Tell the other players someone has disconnected. Remove him from the player list and put his cards in the deck. Chat needs to display that the Player has disconnected.
E008	Server	Client	Invalid input was provided for the username (amount of characters is limited, character types cannot be delimiter).	"Invalid Username. Please enter a valid username according to specifications."	Ask the client to re-enter the Player Name.
E009	Server	Client	The chat message could not be sent.	"The message could not be sent."	Tell the client that the message could not be sent.
E010	Server	Client	The card selected is invalid for the requirements of the table (current Color and current Value)	"Please select a valid card for the game."	Request input until a valid card number is entered or card drawn.
E011	Server	Client	Can not add computer players more than the capacity	“You have requested more computer players than there are spaces available.”	Request another input.

Relevant Design Choices:

The errors were split up to ensure specificity, which will enable more robust processing. If all errors are processed correctly, the game can outlast many different situations that could create crashes.

Useful Notes when Using the Protocol:

Almost all the protocol has been covered extensively with the tables provided above. A Flow Diagram will follow this section of writing to solidify the concepts discussed already. It is important to note that the BGI protocol message sends the latest version of the game to all clients. Where the processing of the validity of a client's move takes place is subject to the development of the game, and should be done through best practices. However, it is important to note that implementations need to account for a way in which the game runs smoothly in sporadic client behavior, such as inactivity/disconnected players. As there are many ways to account and prevent these issues, it is again recommended to use best practices wherever possible, and the design of the protocol does not necessarily impair mechanisms to combat these issues. Rather, the protocol provides a basic template that can be built upon to allow for more convenience and integrability with various game designs so long as the rules of the protocol are respected.

Flowchart for Protocol Correspondences:

Diagram 1:

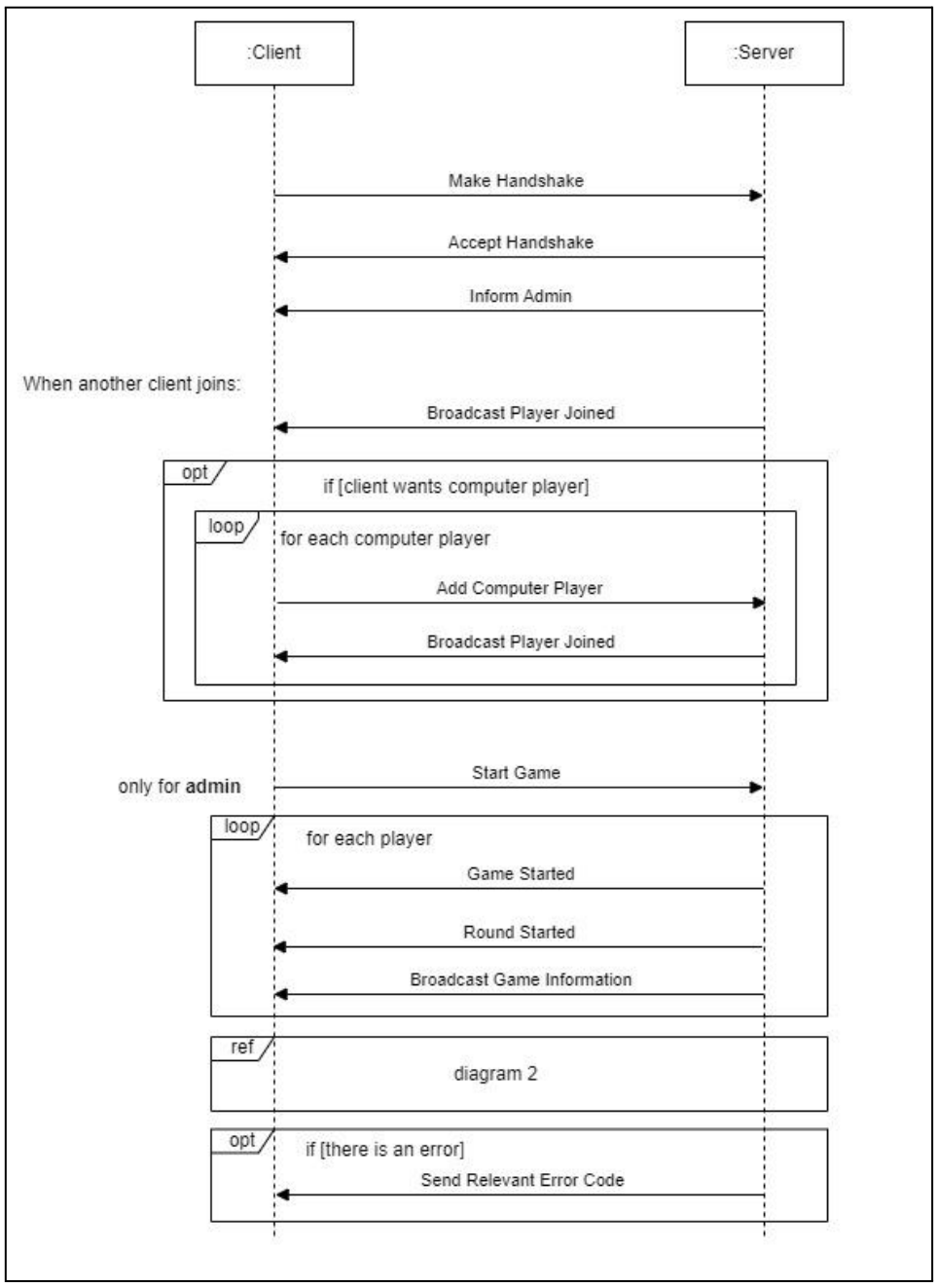


Diagram 2:

(shows the flow of the game, once started)

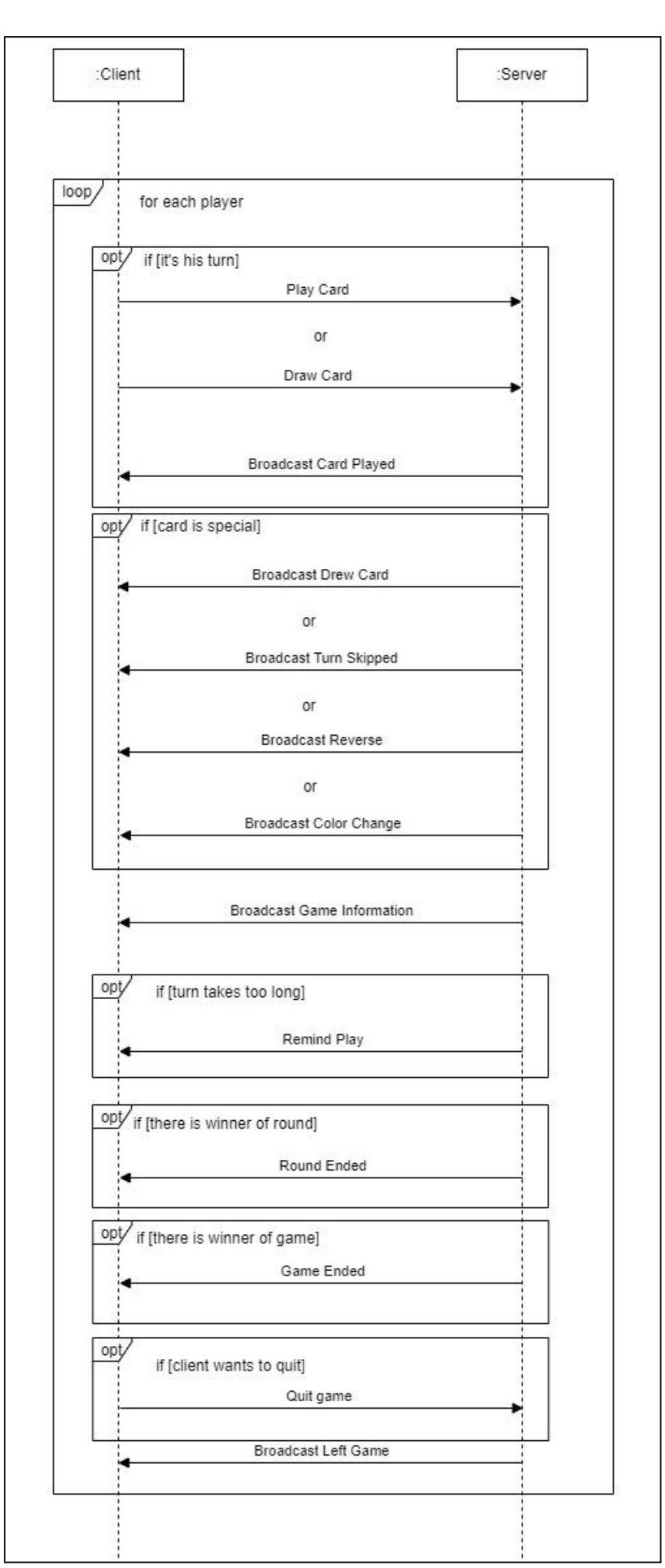
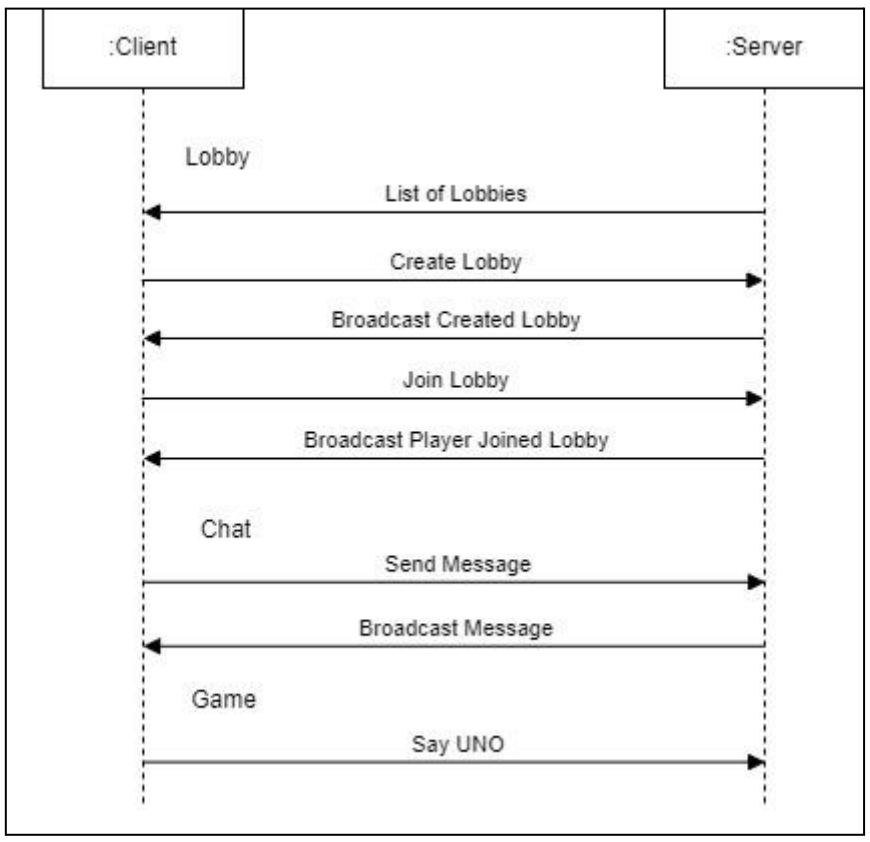


Diagram 3:



Protocol Design Choices and Justification:

Addressing the fact that there is a large number of protocol messages and codes, this was done to ensure that the server and client processing accounts for as many cases as possible. Separating commands and actions was done so that there is less ambiguity in the implementation of the protocol, and as much specificity as possible to ensure that the protocol structure is respected and that compatibility is ensured. This being said, it was necessary to ensure that protocol codes were diverse and specific to a particular task, correspondence, or event. A large number of error codes was required to ensure greater specificity in error handling, ensuring that the game responds to as many areas of concern in distinct ways that it can. Furthermore, in relation to the Java files provided, it is important to note that the planning for the protocol was to have smaller, more specific methods that had certain implementations as opposed to large, less robust methods. It was decided to have one entry and one exit point for the client and server each, which then uses other methods to create data to be transferred. In cases where data is received, it is processed properly and the relevant methods are called to ensure that data processing remains robust. This enabled the creation of a protocol that was organized, methodical, and suited the needs of the Uno Game. The protocol was devised as closely related as possible to the execution of the Uno Game itself (accounting for different game-modes as well), and so long as both the rules of the game and the guidelines of the protocol are respected, then compatibility will not be an issue in implementation.

Feature Completeness:

As seen below the protocol can be used to achieve all the mandatory and additional features required in this project. Refer to the linked sections for more detailed description and format guidance.

Required Features:

1. **You should be able to play UNO with at least 2 people over a network with a client & server application.**
   - This requirement is fulfilled by the Server to Client and Client to Server tables seen above, refer to the tables for the description of each code and examples on how to use them.
2. **The client should be able to connect to a server, play a game and announce the winner.**
   a. **Connect to a server**

      Client to Server: MH

      Server to Client: AH

   b. **Play a game**

      Client to Server: SG, PC, DC, LG

      Server to Client: IAD, BPJ, GST, RST, BGI, BCP, BDC, BTS, BRS, BCC, BLG

   c. **Announce winner**

      Server to Client: RE and GE
3. **The server should be able to host at least one game with 2 players, following the rules of UNO and determine the winner**
   - As seen in requirement two using this protocol allows for hosting at least 2 players as the Server has the IAD which makes a player an admin it is possible to start a game (SG) once enough players have joined. With every player that joins (BPJ) is used to broadcast this. Using the error codes and ERR the game rules can be specified and the GE and RE protocols can be used for announcing winners.
4. **Required to have a UI in the client to play a game.**
   - UI for each pair would probably be different, it must be ensured that at the end regardless of user input, the client communicates with the server and vice versa using the format given in the Types, Server to Client and Client to Server tables.
5. **Your application should handle common exceptions and errors correctly**
   - Table errors include Client-Server communication errors that could occur while playing UNO. These are errors that may occur, with invalid input that does not follow the protocol format or for instance trying to play a card that is not possible to play according to game rules. For more details refer to table errors.
6. **You should have a computer player**
   - The protocol allows for adding a computer player once a client is connected to the server and is made admin by the server (IAD). The protocol code for this is:

     Client to Server: ACP

Additional features:

1. **A chat function where players can send messages to each other**

    Client to Server: SM

    Server to Client: BM

2. **Team-play, add the option to play a game up to 10-players**

    - For this functionality the protocol supports players joining the game until the admin starts the game. If the players in the game are at capacity then E005 (Server to Client: ERR) can be used to indicate this to players that still want to join the game.

3. **Multi-game server, the server supports hosting multiple games simultaneously with different players.**

    - This functionality can be realized with the protocol, only limitation is if you want to implement it.

4. **A lobby, where clients can join and decide together to play a game of UNO, instead of the server creating the games**

    - After a handshake is made with the server and is accepted, the following protocols can be used to realize this feature:

    Server to Client: LOL, BCL, BJL

    Client to Server: CL, JL

5. **Saying UNO**

    Client to Server: UNO

6. **Remind player to play when the player is inactive**

    Server to Client: RP

7. **Different game modes (normal, progressive, jump-in, seven-o)**

    - For this feature the SG protocol can be used which is in the Client to Server table by specifying the game mode while communicating the client can choose to start the preferred game mode. Only the admin can start the game (SG).

Conclusion:

The entire document, along with the provided Java Code has been sufficient in informing users of this protocol of the functionality it possesses. It was a collective process to make this project and it required a great deal of teamwork, critical thinking, and coordination.