PR6 Project – Detailed protocol

crochets ne font PAS partie du message

1 byte ⇒ $0 \leq x \leq 255$

Server | Client

Before the start of the game

AT CONNECTION

[GAMES ␣ n ***] — TCP →

↑ uint_8 (1 byte) nb of games not started (can be 0)

Then n messages:

[OGAME ␣ m ␣ p ***]

↑ uint_8 (1 byte) nb of players already there

uint_8 (1 byte) id number of the game

PLAYER REGISTRATION

OR CREATING A NEW GAME

Player can either register for a game of create a new one

TCP ←

[NEWPL ␣ id ␣ port ***] → create new game

4 chars player's UDP port for private messages

8 chars name of player

OR

[REGIS ␣ id ␣ port ␣ m ***] → join existing game

uint_8 (1 byte) nb of the game to join

4 chars UDP port for private messages

8 chars id of player

Server answers either — TCP →

[REGOK ␣ m ***] if everything good

↑ uint_8 (1 byte) nb of the game joined

↳ if a new game was created, this will be a new number

OR

[REGNO ***] if it didn't work

Possible causes:

→ player is already registered in another game

→ another player took the last spot in the meantime

→ trying to create a new game but there is already the max number that exist

→ trying to join a game that has already started

→ trying to join a game that has not been created yet.

PLAYER INDICATES THEY ARE READY

When player is ready to start

TCP ← [START ***]

! Server must loop through the players for each game until they've all sent START.

↳ then player is stuck and can't send messages to the server until game starts!

→ if player sends START without being registered in a game : IGNORE MESSAGE.

Server | Client

Before sending START, player can send other messages (whether they are registered or not)

TCP ← Ask for maze size

[SIZE? ␣ m ***]

uint_8 (1 byte) id of the game

TCP →

Server answers

SIZE OF MAZE

[DUNNO ***] if game doesn't exist

OR [SIZE! ␣ m ␣ h ␣ w ***]

w: width

h: height

BOTH are uint_16 (2 bytes) in little endian && < 1 000

m: uint_8 (1 byte) id of the game

LIST OF PLAYERS

TCP ← [LIST? ␣ m ***]

uint_8 (1 byte) id of the game

TCP →

Server answers

[DUNNO ***] if game doesn't exist

OR

[LIST! ␣ m ␣ s ***]

s: uint_8 (1 byte) nb of players

m: uint_8 (1 byte) id of the game

then s messages:

[PLAYR ␣ id ***]

id: 8 chars id of the player

LIST OF THE GAMES

TCP ← [GAME? ***]

Server answers the same thing than at the beginning of connection (see top of first page), messages GAMES & OGAME)

Server | Client

At the start of the game

id of game uint_8 (1 byte)

TCP →

[WELCO␣m␣h␣w␣f␣ip␣port ***]

nb of ghosts uint_8 (1 byte)

broadcasting port 4 chars

size of maze uint 16 (2 bytes) little endian <1000

broadcasting IP 15 chars completed by # if the IP is smaller

Then to each player.

[POSIT␣id␣x␣y ***]

id of player 8 chars

row & column nb coded as a string on 3 bytes, completed with 0 on the left if needed.

ex row 11 → "011"
col 2 → "002"
row 555 → "555"

Player moves

TCP ←

UP [UPMOV␣d ***]

DOWN [DOMOV␣d ***]

LEFT [LEMOV␣d ***]

RIGHT [RIMOV␣d ***]

distance, on 3 bytes as a string

"055" for 55
"123" for 123
"001" for 1

Server answers

TCP →

[MOVE!␣x␣y ***]

new position string on 3 bytes

gives 10 points

or if a ghost was caught

ex: "001", "012", "123"

[MOVEF␣x␣y␣f ***]

player's score, on 4 bytes as a string

ex: "1234", "0012", "0002"

if a ghost was caught, server also sends

UDP multicast →

[SCORE␣id␣p␣x␣y ++++]

player id (exactly 8 chars)

player's nb of point on string 4 bytes

point of ghost when caught

Player quits

TCP ←

[QUIT ***]

Server answers

TCP →

[GOBYE ***]

Server | Client

Get list of players

TCP ←

[GLIS?***]

Server answers:

TCP →

[GLIS! u s ***]

↑ nb of players, uint8 (1 byte)

then s messages:

[GPLYR uid u x u y u p ***]

- uid: string of 8 chars
- x: position, string of 3 chars, "001", "012", "123"
- y: position, string of 3 chars, "001", "012", "123"
- p: score, string of 4 chars, "0001", "0101", ...

If player sends a message but game has ended

Server sends

[GOBYE***]

and ends the connect0

When ghosts move

UDP multicast →

[GHOST u x u y +++]

↑ position, string on 3 bytes ex "001", "012", "123"

Endgame

Game ends when there are no more ghosts

UDP multicast →

[ENDGA u id u p +++]

- id: player who won
- p: their score

If 2 players have the same amount of point, winner is the first one found in the list of players

If no players left in game: game ends but no message is sent.

Server | Client

Public message

TCP ←

[MALL? ␣ mess ***]

string of max 200 chars, doesn't contain *** or +++

Server answers TCP →

[MALL! ***]

then multicasts the message

UDP multicast →

[MESSA ␣ id ␣ mess +++]

8 chars; max 200 chars

Private message

TCP ←

[SEND? ␣ idu ␣ mess ***]

recipient's id (8 chars) exactly

string, max 200 chars (doesn't contain *** or +++)

Server answers TCP →

[SEND! ***] if OK

OR

[NSEND ***] if problem

Possible problems are:

→ id player doesn't exist

→ id player is not in the game (or has left the game)

Then server sends the message UDP →

[MESSP ␣ id ␣ mess +++]

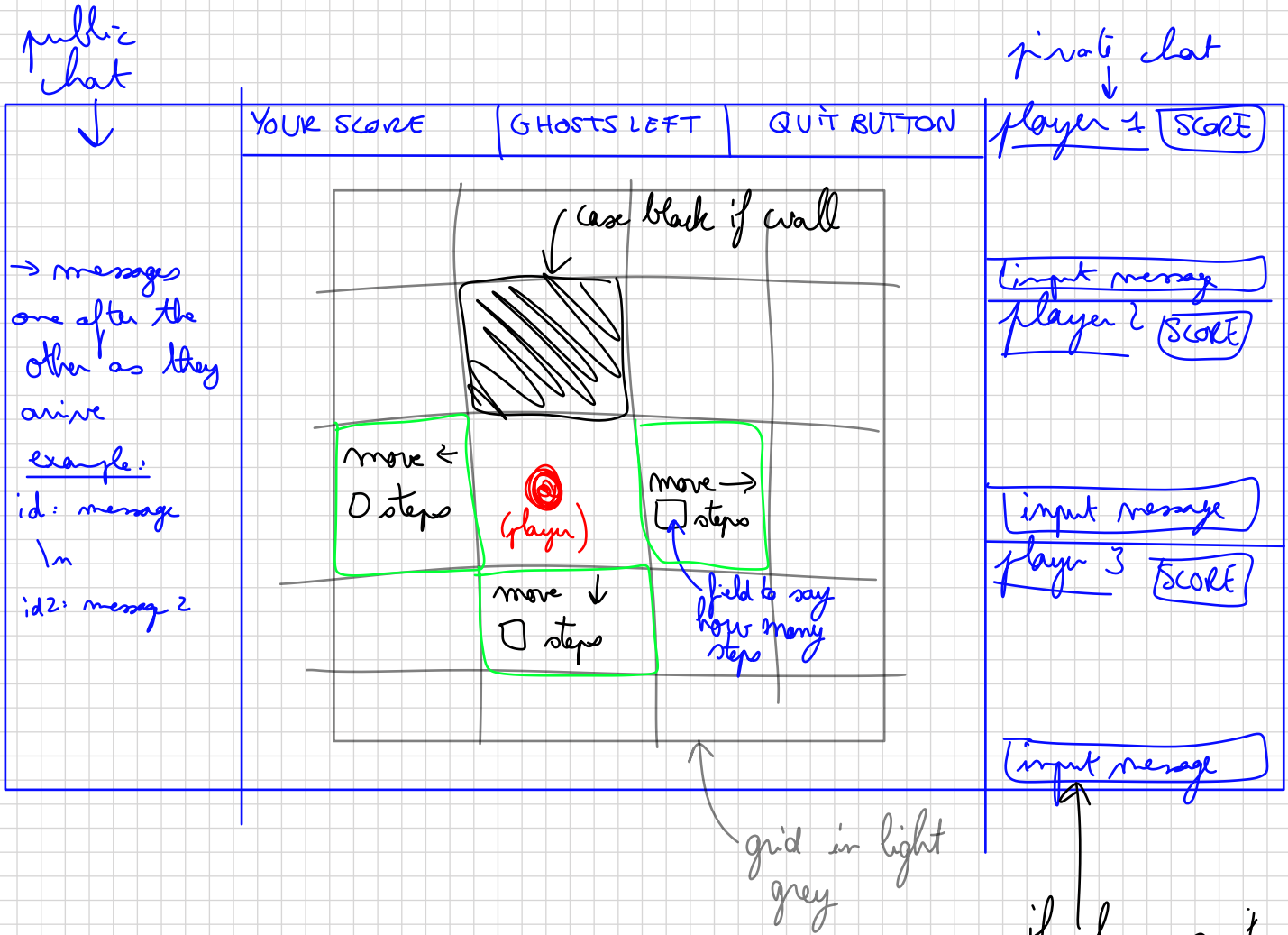
Recipient doesn't acknowledge

id of player who SENT the message (exactly 8 chars)

string, max 200 chars, doesn't contain *** or +++

Client interface proposal

GAME



We decide MAX 4 players / game

REGISTRATION