Programação de Sistemas – 24/25
Project report

João Lopes 99973
Carolina Tilley 99899

1 Introduction

This project explores the ZMQ library and parallel implementations (forks and threads). It aims to develop a functioning space invaders game using ZMQ communications between server and clients.

2 Implemented functionalities

2.1 Part A

Table 1: Implemented functionalities (PART A)

	Not implemented	With faults	Totally Correct
game-server.c			
Suitable sockets			x
Assignment of Astronaut letters			x
Storage of clients Astronauts and Aliens			x
Management of astronauts			x
astronaut-client.c			
Connect			x
Movement			x
Zap			x
Disconnect			x
Display score			x
outer-space-display.c + display on game-server.c			
Correct update of screen			x
Zapps			x
Destruction of aliens			x
Rules			
Movement of Aliens			x
Astronaut zapping			x
Astronaut zapping (fire rate)			x
Astronaut zapping (0.5 second ray)		x	
Aliens are destruction (points)			x
Astronauts stunning			x
Misc			

Messages validation		x	
Optimization of communication			x
Code organization		x	
Functions return values validation	x		

2.2 Part B

Table 2: Implemented functionalities (PART B)

	Not implemented	**With faults**	**Totally 1 Correct**
game-server.c			
2 Suitable sockets			x
3 Disconnect of clients			x
4 Communication thread			x
5 Aliens thread			x
6 Synchronization			x
astronaut-display-client.c			
Connect			x
Movement			x
Zap			x
Disconnect			x
Update of display			x
Display score			x
Threads			x
space-high-scores			
Client on other language			x
Protocolbuffer messages			x
Rules			
Movement of Aliens			x
Astronaut zapping			x
Astronaut zapping (fire rate)			x
Astronaut zapping (0.5 second ray)		x	
Aliens destruction and points			x
Astronauts stunning			x
Aliens population recovery			x

Misc			
Messages validation		x	
Optimization of communication			x
Code organization		x	
Functions return values validation	x		

2.3 Description of faulty functionalities

Astronaut zapping (0.5 second ray): this functionality is implemented in both Part A and Part B. However, every second we also clear the board. Therefore, occasionally the zapping effect is not visible during the 0.5 seconds.

Message Validation: when quit, no acknowledgments are received or sent.

Code organization: while the code has been divided into several .c and .h files, the overall structure still lacks optimal organization and clarity.

Functions return values validation: although we validate the return of some functions, this is not consistently applied throughout the entire code.

3 PART A Description of code

3.1 Data types

Structs:

- Player[8] //stores all the active players' information
    - int score //stores the score of the player
    - int position[2] //stores the player position
    - int fire //flag for zapping
    - int stun //flag for stun
    - time_t start_time_fire //last time player fired
    - time_t start_time_stun //last time player was stunned
    - char character //player character

Arrays:

- act_player[8] // stores all the active players
- grid[20][20] //stores all the aliens ->0 for blank space, 1 for occupied by aliens

3.2 Functions List

- game-server.c
    - init_pl //initializes player data and active player arrays
    - draw_grid //draws the borders of a grid window

- draw_numbers //displays column numbers at the top and row numbers on the side of a grid
- populate_aliens //randomly places aliens (‘*’) on the grid within a restricted area and updates the display
- update_score //updates and displays the scores of players in a score window
- fork_aliens //manages the communication and movement of aliens by sending periodic messages to a father server
- prepare_message_display //prepares a message display structure for communication
- find_ch_info // finds the index of a character in a player array
- new_position //updates the position based on a given letter and direction
- seconds_passed //checks if a specified amount of time (in seconds) has passed since a given start time
- draw_zap_line //draws a zap line (vertical ou horizontal) on the grid based on the player’s character type
- move_aliens //moves aliens randomly on the grid while ensuring valid positions and avoiding overlaps
- draw_aliens //draws aliens (‘*’) on the grid window and checks if any aliens remain active. Displays ‘GAME OVER!’ if no aliens are left

- astronaut-client.c
  - no additional functions were implemented (simple read key like in lab)

- outer-space-display.c.
  - init_variables// initializes variables received from message to local arrays

  The following functions were already described in ‘game-server.c’:
  - draw_grid
  - draw_numbers
  - draw_aliens
  - draw_players
  - draw_zap_line

- update_score
- find_ch_info
- new_position
- seconds_passed
- move aliens

3.3 Implementation details

- Zapping fire rate

  Each player has a zapping flag. The first time they zap the flag is set to 1 and a timestamp is acquired. All the following zap actions are only processed if the timestamp comparison allows the zap to occur (criteria is if time difference is greater than 3)

- Description of the implementation of the astronaut zapping 0.5 second ray display

  Each time a player zaps, if the zap is legal, a line is drawn in the map, then a 0.5 while loop (stuck) occurs

- Player Stun

  Each player has a stun flag. The first time they get stunned the flag is set to 1 and a timestamp is acquired. All the following movement and zap actions are only processed if the timestamp comparison allows the zap to occur (criteria is if time difference is greater than 10). To turn the flag to 1 when zapping, we check if the zap hits any character.

4 PART B Description of code

4.1 Data types

To store the aliens no additional struct was created.

4.2 Functions List

In this section students should present a list with new of modified functions from Part A to Part B:

- game-server.c
  - communication_thread //manage all the player communication and game mechanics
  - move_aliens_thread //manage the movement of aliens and the alien population recovery
  - keyboard_listener ///manage quitting and sending quit messages to all other threads

Score buffer generated functions: score_update__get_packed_size, score_update__pack.

- astronaut-display-client.c
  - astronaut_client_thread //manage astronaut movement
  - display_thread //manage display of outer space
- space-high-scores.py
  - space highscores recipes an array with the player scores and another with the active players prints all scores of the active players

4.3 Implementation details

- Description of the implementation of the astronaut zapping fire rate.

  Same as in section A
- Description of the implementation of the astronaut zapping 0.5 second ray display.

  Same as in section A
- Description of the implementation of the astronauts stunning (immobility).

  Same as in section A
- Description of the implementation of the aliens destruction and points

When a player zaps all the aliens (‘*’) he passes are destroyed both in the screen and in grid[20][20]. For every alien (‘*’) destroyed, a point is awarded.

- Description of the implementation of the aliens population recovery

In a thread function, every 10 seconds we compare the number of aliens to the previous number of aliens. If the number remains the same we increase the number of aliens by a factor of 0.1.

4.4 Threads

In the section students should present a list of every thread implemented, divided by components:

- astronaut-display-client.c
  - astronaut_client_thread //manage astronaut movement
  - display_thread //manage display of outer space

- game-server.c
  - communication_thread //manage all the player communication and game mechanics

    - move_aliens_thread //manage the movement of aliens and the alien population recovery
    - keyboard_listener //manage quitting and sending quit messages to all other threads

- astronaut-client.c
    - message_receiver // is created to read the termination message from the game-server

4.5 Shared variables

- Global variables:

  grid[20][20]

- Shared variables:

  aliens thread

  -pub_socket //for sending the new map update to all the displayed

  -grid_win //to update the window

  -ch_data //all the players and their data

  -act_pl // all the active players

  communication thread

  -req_socket, pub_socket // for normal communication

  -grid_win,score_win // for display

  -char_data,act_player// for data management and game behaviour

  termination_thread

  -pub_sucket // for sending termination message for display clients

  -contex // for creating REQ communication with non display clients and send terminate message

4.6 Synchronization

- grid[20][20]
  because it is a global variable and it is accessed by both aliens thread and communication thread grid as a lock and unlock mutex, in game-server.

- quit flag
  from the astronaut-display-client for correct thread termination.

5 PART A Communication

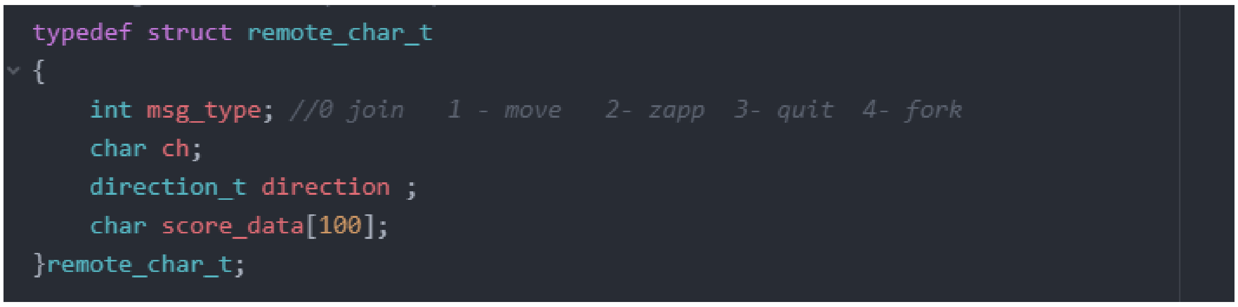
5.1 Sockets

- game-server.c
  - rep socket \\ for astronaut clients
  - pub socket \\ for display clients
  - fork socket \\ created by the fork to sends messages to the “father” every 1 second

- outer-space-display.c
  - sub socket \\ for sending display messages
- astronaut-client .c
  - req socket \\ for receiving client messages

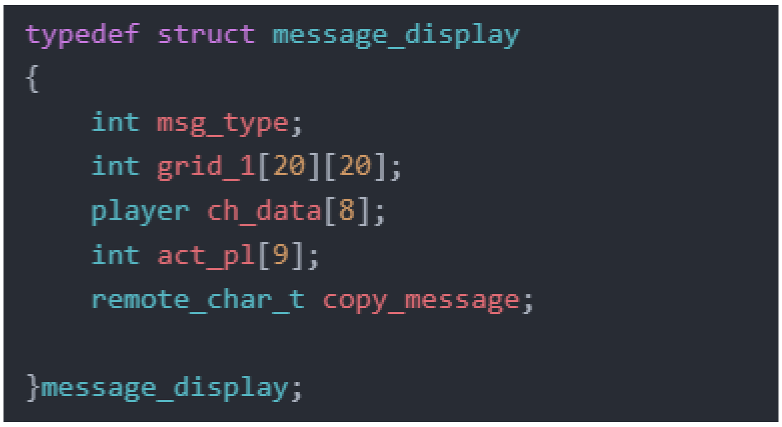
5.2 Transferred data

- For req/rep communication



```c
typedef struct remote_char_t
{
    int msg_type; //0 join   1 - move   2- zapp  3- quit  4- fork
    char ch;
    direction_t direction ;
    char score_data[100];
}remote_char_t;
```

- For pub/sub



```c
typedef struct message_display
{
    int msg_type;
    int grid_1[20][20];
    player ch_data[8];
    int act_pl[9];
    remote_char_t copy_message;

}message_display;
```

sends grid, player and copy of the current message received

5.3 Error treatment

Check if the player exists.
Check if the player is active and the maximum number of players is not yet full.

6 PART B Communication

6.1 Sockets

- Game-server.c
  - rep socket \\ for astronaut clients
  - pub socket \\ for display clients
  - termination \\socket that sends astronaut clints termination
  - req \\sends communication thread termination

- outer-space-display.c
  - sub socket \\ for receiving display messages
- astronaut-client .c
  - req socket \\ for sending messages
  - rep socket \\ for receiving termination message
- astronaut-display-client .c
  - req socket \\ for sending messages
  - sub socket \\ for receiving display messages

6.2 Transferred data

No changes were made.

6.3 Error treatment

Check if the player exists.
Check if the player is active and the maximum number of players is not yet full.

7 Conclusion

In this project, we learned a lot about ZeroMQ (ZMQ) as a library. It was nice to see how easy and seamless it integrates client and message management compared to other select-based communication programs we developed in previous disciplines.

In terms of implementation, the hardest part in Part A was handling fork communication and implementing the 0.5-second zap timer. In Part B, most of the work focused on migrating the functions to threads and ensuring they all terminate (join) cleanly.