Systems Programming
2023/2024
Project Assignment – Part B

Group 5 (95581 & 95686) Report

Implemented Functionalities

Lizard Client:

1. Lizard Connect:
    a. The lizard client sends a message to the server with a connect request;
    b. The lizard client receives a response containing the lizard's identifier and a nonce.
2. Lizard Movement:
    a. The user presses one of the arrow keys to move;
    b. The lizard client sends a message to the server containing the lizard's identifier, the movement direction, and the nonce;
    c. The lizard client receives a message confirmation and a new nonce.
3. Lizard Disconnect:
    a. The user presses the 'Q' key;
    b. The lizard client sends a message to the server with the disconnect request, the lizard's identifier, and the nonce;
    c. The lizard client receives a message confirmation;
    d. The lizard client closes.
4. Board display:
    a. When the client connects it receives the current state of the board and displays it along with other lizards' scores in a ncurses window
    b. Every time there is an update to the board the lizard client receives it and displays the updates on the board.

Roaches Client:

1. Roaches Connect:
    a. The user has the option to select the number of roaches to be controlled, if they don't a random number will be chosen.
    b. The roaches client sends a message to the server with a connect request;
    c. The roaches client receives a response containing the roaches' identifiers and nonces.
2. Roaches Movement:
    a. A random roach and a random movement direction are selected;
    b. The roaches client sends a message to the server containing the roach's identifier, the movement direction, and the nonce;
    c. The roaches client receives a message confirmation and a new nonce for the roach.

3. Roaches Disconnect:
    a. The user terminates the program;
    b. The roaches client catches the signal;
    c. The roaches client sends a message to the server with the disconnect request, the roachs' identifiers, and their nonces;
    d. The roaches client receives a message confirmation;
    e. The roaches client closes.

Wasps Client:

Identical to the roaches client.

Server:

Receives all previous mentioned messages from the clients, executes the game's logic and responds to the clients. Furthermore, every time the board state changes, the server broadcasts that change to all the lizard clients.

Error treatment / Cheating:

Sensible functions' outputs are checked and handled accordingly.

To prevent cheating we use nonces to ensure messages are signed and fresh. Upon connection each lizard, roach, and wasp is attributed both an identifier and a nonce. With every request the clients send an identifier and the corresponding nonce, the server only executes the request if the nonce matches. In the response the server includes a new nonce.

To ensure proper synchronization due to the implementation of threads, mutex were used to lock critical sections.

Architecture of the System

Lizard Client:

The lizard client connects to the server with a requester-type ZeroMQ socket that sends the connect, movement, and disconnect messages, and a subscriber-type ZeroMQ socket to receive field updates from the server.

Roaches Client:

The roaches client connects to the server with a requester-type ZeroMQ socket that sends the connect, movement, and disconnect messages.

Wasps Client:

The wasps client is identical to the roaches client.

Server:

The server maintains three arrays that store all of the information required to run the game:

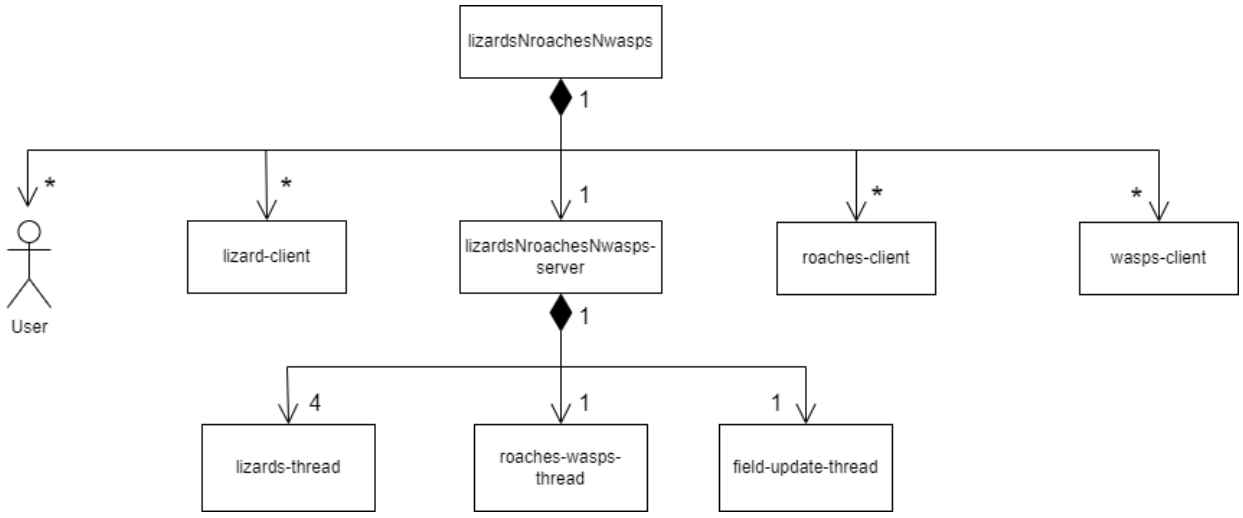
- ‘lizard_data’ stores all the information about the lizards, namely, identifiers, position on the board, scores, tail size and location, and nonces.
- ‘roaches_data’ stores all the information about the roaches, namely, identifiers, position on the board, value, and nonces.
- ‘wasps_data’ stores all the information about the wasps, namely, identifiers, position on the board, and nonces.

The server utilizes a ZeroMQ router-dealer request-reply pattern to answer lizard client requests because it launches 4 threads to handle client requests. The main thread has a ZeroMQ router socket to receive requests, this socket transmits the requests to the ZeroMQ dealer socket, also in the main thread, the dealer then distributes these requests to the responder sockets in each of the 4 “worker” threads. After processing the request and executing the logic, each thread sends the reply to the dealer who in turn sends it to the router, who finally delivers it to the client.

To answer roaches and wasps clients’ requests, the server launches one thread that has a ZeroMQ responder socket that simply receives the requests, executes the game’s logic, and responds to the clients.

For field updates, the server uses a ZeroMQ fan-in pattern. Each lizard and roach-wasp “worker” thread has a publisher socket that they use to send field updates. The server launches a thread that has both a subscriber and a publisher socket. This thread uses the subscriber socket to receive field updates from the lizard and roach-wasp “worker” threads and update the server’s board, it then utilizes the publisher socket to send the same field updates to the lizard clients.

Below we show a block diagram illustrating the architecture of the system.

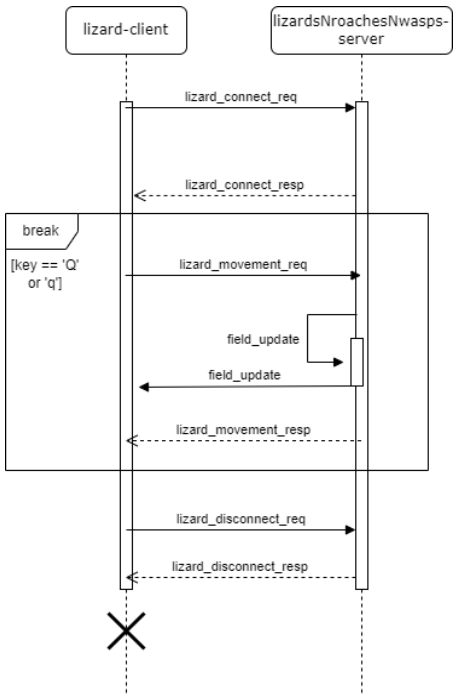


Communication Protocols

Lizard Client - Server:

1. Lizard Connect:
    a. Lizard client sends a LizardConnectReq message to the server;
    b. The server responds with a LizardConnectResp message that contains whether or not the request was successful, the character attributed to the lizard and the nonce.
2. Lizard Movement:
    a. Lizard client sends a LizardMovementReq message to the server, containing the character of the lizard, the direction to where it wants to move and the nonce;
    b. The server responds with a LizardMovementResp message that contains whether or not the request was successful, the updated score of the lizard and the new nonce.
3. Lizard Disconnect:
    a. Lizard client sends a LizardDisconnectReq message to the server, containing the character of the lizard and the nonce;
    b. The server responds with a LizardDisconnectResp message that contains whether or not the request was successful.
4. Field Update:
    a. Whenever there is a field update, the lizard and roach-wasp “worker” threads send a FieldUpdate message to the field_update thread, containing the character to be drawn, the coordinates, and, if there is a lizard score update, the character of the lizard and the updated score;
    b. When the thread receives this message, it updates the server’s board and forwards the exact same message to all the lizard clients;
    c. When a lizard client receives a FieldUpdate message, it updates its board and, if necessary, the lizards’ scores.

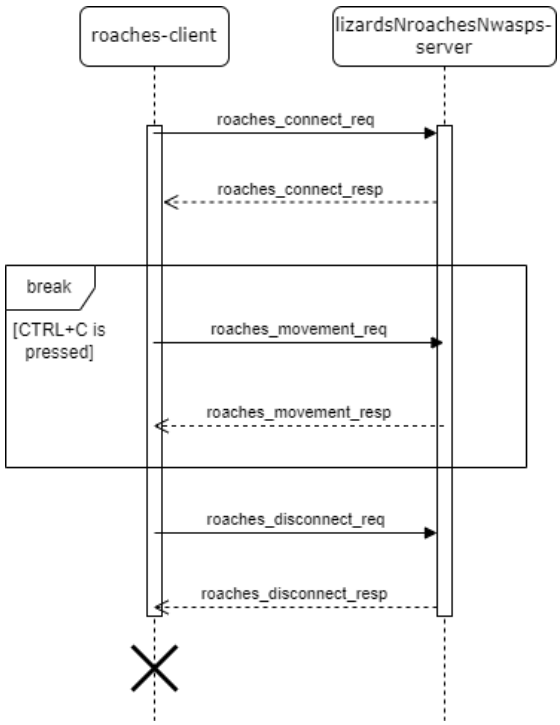
Below we show an interaction diagram illustrating the exchange of messages between the lizard clients and the server.



Roaches Client - Server:

1. Roaches Connect:
    a. Roaches client sends a RoachesConnectReq message to the server, containing the number of roaches it wishes to control and their respective scores;
    b. The server responds with a RoachesConnectResp message that contains whether or not the request was successful, the identifier attributed to each roach and the nonce of each roach.
2. Roaches Movement:
    a. Roaches client sends a RoachesMovementReq message to the server, containing the identifier of the roach, the direction to where it wants to move and the nonce;
    b. The server responds with a RoachesMovementResp message that contains whether or not the request was successful and the roach's new nonce.
3. Roaches Disconnect:
    a. Roaches client sends a RoachesDisconnectReq message to the server containing the number of roaches it controls, their identifiers and their nonces;
    b. The server responds with a RoachesDisconnectResp that contains whether or not the request was successful.

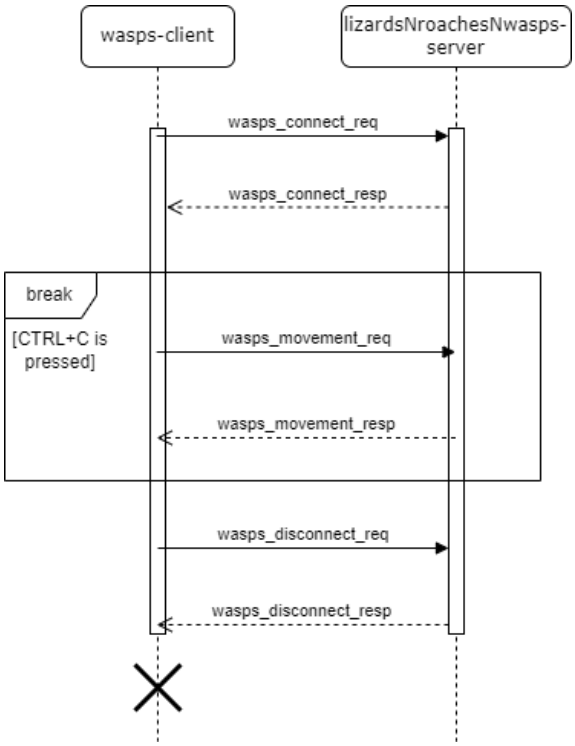
Below we show an interaction diagram illustrating the exchange of messages between the roaches clients and the server.



Wasps Client - Server:

1. Wasps Connect:
    a. Wasps client sends a WaspsConnectReq message to the server, containing the number of wasps it wishes to control;
    b. The server responds with a WaspsConnectResp message that contains whether or not the request was successful, the identifier attributed to each wasp and the nonce of each wasp.
2. Wasps Movement:
    a. Wasps client sends a WaspsMovementReq message to the server, containing the identifier of the wasp, the direction to where it wants to move and the nonce;
    b. The server responds with a WaspsMovementResp message that contains whether or not the request was successful and the wasp's new nonce.
3. Wasps Disconnect:
    a. Wasps client sends a WaspsDisconnectReq message to the server containing the number of wasps it controls, their identifiers and their nonces;
    b. The server responds with a WaspsDisconnectResp that contains whether or not the request was successful.

Below we show an interaction diagram illustrating the exchange of messages between the wasps clients and the server.



Changes between the two versions of the project:

Overall changes:

The messages exchanged between server and clients are now all specified in a proto file.

Lizard Client:

The lizard client maintains the same logic but with the display app integrated into it.

Roaches Client:

The roaches client is the same but now when it is terminated it sends a message to the server who erases all the corresponding roaches.

Wasps Client:

The wasps client was only added in the second part.

Server:

The server changed the most. Instead of only one thread for everything, now there are 4 threads to handle lizard clients' requests, one thread to handle roaches and wasps clients' requests, and one thread to draw the board and send field updates. When any client disconnects, all the identifiers associated with them are erased from the board. Lizards can now get a negative score and when they do their tail is no longer drawn.