

Systems Programming

2023/2024

Project Assignment – Part B

 **lizardsNroachesNwasps**  

In the second part of the project, students will continue the implementation of the simple variation of the snake game, where lizards walk in a field and eat cockroaches. In this new version of the game, Asian wasps will also exist.

The server manages the field, while lizards, cockroaches and wasps are controlled by different programs/clients that send movement indications to the server.

In this new version of the game, the applications that display the field in progression throughout the game will be merged with the lizard controllers.

1 lizardsNroachesNwasps

In the **lizardsNroachesNwasps** game, users control a lizard that moves in a field, eats cockroaches, and is stung by wasps. In those fields, cockroaches (represented by a digit) and wasps (represented by #) move randomly.

Each lizard score varies as follows:

- When a lizard rams another one, the health of both lizards is equalized to the average of their respective scores.
- If a lizard eats a cockroach, its score increases by the value of the ingested cockroach.
- If a lizard is stung by a wasp, its score decreases by 10 points.

Although lizards have a long body, the previous interactions only affect the head of the lizard.

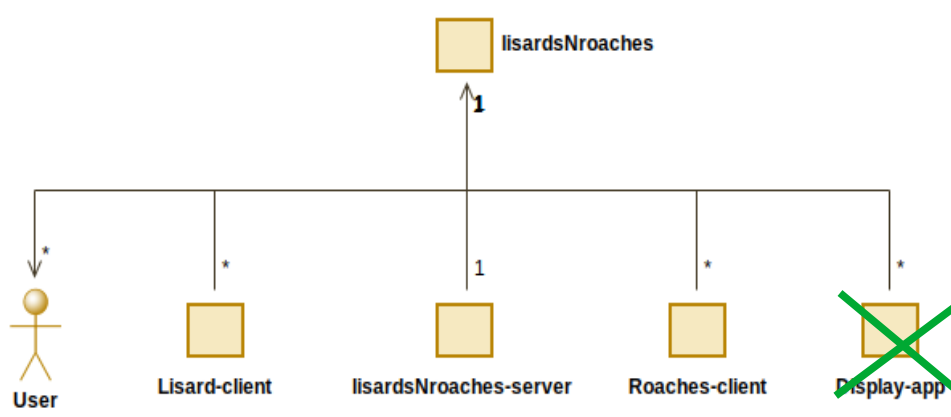
A lizard wins the game when it reaches 50 points.

A lizard loses when it reaches a negative score.

2 Project Part A

In the second part of the project, students will implement the **lizardsNroachesNwasps** game using a distributed architecture, where each lizard is controlled by a user, the cockroaches are controlled by roaches' bots (a different kind of client), and wasps are controlled by the wasps' bots (a different kind of client):

- **Lizard-client** is an application that reads the keys pressed by the user, sends them to the server, receives the user score and shows the board with all the participants in the same way as the **lizardsNroachesNwasps-server**.
- **lizardsNroachesNwasps-server** is a server application that receives messages from all the clients, handles the board/field (lizards and cockroaches), and sends updates to the **Lizard-client**. This application shows the board with all the participants.
- **Roaches-client** is an application that sends random movements for a set of up to 10 roaches.
- **Wasps-client** is an application that sends random movements for a set of up to 10 wasps.
- The **Display-app** will be removed from this game and will be integrated into the **Lizard-client**.



Several users can play simultaneously (launching various **Lizard-client**), and multiple **Roaches-client** or **Wasps-client** can control their set of cockroaches and wasps, respectively.

The **Lizard-client**, **Roaches-client** and **Wasps-client** are independent processes/clients that interact with the server using **ZeroMQ TCP sockets**.

Every lizard should be identified by a unique letter. This letter should be assigned by the server when the **Lizard-client** first connects.

2.1 Interaction

The server waits for messages from the clients and depending on the received message changes its state and replies to the client accordingly.

The basic messages exchanged between the various components of the game are:

- **Lizard_connect + response** (from the **Lizard-client** to the server)
- **Lizard_movement + response** (from the **Lizard-client** to the server)
- **Roaches_connect + response** (from the **Roaches-client** to the server)
- **Roaches_movement + response** (from the **Roaches-client** to the server)
- **Wasps_connect + response** (from the **Wasps-client** to the server)

- **Wasps_movement + response** (from the **Wasps-client** to the server)
- **Disconnect + response** (from **Lizard-client** to the server)
- **Disconnect + response** (from **Roaches-client** to the server)
- **Disconnect + response** (from **Wasps-client** to the server)
- **Field_update** (from the server to the **Lizard-client**)

Every **Lizard-client** needs to send a **Lizard-connect** message at startup. The server stores the relevant client and lizard information in a list/array and replies to it. Whenever the **Lizard-client** sends a **Lizard-movement**, the server should update the board, update the scores, reply to the client, and send a **Field_update** message to all the **Lizard-client**'s.

Roaches-clients should also connect to the server before starting to control its roaches. The **Roaches-client** decides on the roaches' movements and sends them to the server (**Roaches_movement** message). The server updates the board and sends a message to all the **Lizard-client**'s.

Wasps-clients should also connect to the server before starting to control its wasps. The **Wasps-client** decides on the wasps' movements and sends them to the server (**Wasps_movement** message). The server updates the board and sends a message to all the **Lizard-client**'s.

If multiple **Lizard-client**'s are connected, the server will process various **Lizard_movement**. Every time the field or any score changes (due to movements) the server should send a **Field_update** to all the **Lizard-clients**.

2.2 lizardsNroachesNwasps-server

The **lizardsNroachesNwasps-server** is a C program that uses **ZeroMQ TCP sockets** to interact with the other game components.

The maximum number of simultaneous players is twenty-six (26). Students should decide what happens to a client that sends a **Lizard_connect** message when 26 clients are already connected.

The sum of all cockroaches and wasps cannot occupy more than 1/3 of the field. Students should decide what happens if a **Roaches-client** or **Wasps-client** tries to connect and there already are enough cockroaches and wasps on the board.

The server should store all the clients and all the relevant information (e.g., player position, health, ...) in lists or arrays. A client (**Lizard-client**, **Roaches-client** or **Wasps-client**) is inserted into a list or array when the server receives a **Lizard_connect** or **Roaches_connect** or **Wasps_connect** messages and is removed when the server receives a corresponding disconnect (**Lizard_disconnect**, **Wasps_disconnect** or **Roaches_disconnect**) message.

The server should be multi-threaded:

- 4 threads for handling messages from all the **Lizard_client**'s.
- one thread for handling messages from the **Roaches-client**'s or **Wasps-client**'s.

2.3 Lizard-client





The **Lizard-client** is a C program that interacts with a server using **ZeroMQ TCP sockets**. This program allows a user to control a lizard on the field.

The address and port of the server should be supplied to the program as a command line argument.

This program reads cursor keys presses from the keyboard and forwards them to the server (messages **Lizard_movement**) to move the user lizard.

Before sending **Lizard_movement** messages, the **Lizard-client** should connect to the server and receive the assigned letter. Only after receiving this message the client goes into the loop that:

- reads a key press;
- sends the respective **Lizard_movement** message to the server;
- receives a reply with the Lizard score;

The lizards are controlled using the cursor     keys:

If the user presses the **q** or **Q** keys, the client should terminate and send a **Disconnect** message to the server.

The new version of this program mirrors the content displayed by the server on its screen and is updated each time there is a movement by lizards, cockroaches or wasps. Students should reuse the code of the old **Display-app** into the new version of the **Lizard-client**.

2.4 Roaches-client

The **Roaches-client** is a C program that interacts with a server using **ZeroMQ TCP sockets**. This program controls some cockroaches in the field.

The address and port of the server should be supplied to the program as a command line argument.

This client randomly decides the movement (up, down, left, right) of each one of its roaches and sends this information to the server using the **Roaches_movement** message.

To make this movement realistic, the period between cockroaches' movements should not be fixed, and not all cockroaches should move at the same time.

Each **Roaches-client** can control between 1 and 10 cockroaches.

Each roach has a score between 1 and 5 that is randomly defined by the **Roaches-client** for each of its roaches at startup. This value is used to draw the cockroach and is given to the lizards that eat cockroaches.

When this application terminates, the server should receive a **Roaches_disconnect** message and remove the corresponding roaches.

2.5 Wasps-client

The **Wasps-client** is a C program that interacts with a server using **ZeroMQ TCP sockets**. This program controls some wasps in the field.

The way this program works is exactly the same as the way the **Roaches-client** works, but for wasps.

2.6 Display-app

This application should be removed from the project.

The **Display-app** source code should be integrated into the **Lizard-client** so the user sees the field during the game.

2.7 User interfaces

2.7.1 Lizard-client

The **Lizard-client** should implement a simple NCurses interface to allow reading the cursor keys and print all the lizards scores.

The new version of this application should show the game field (as the server and old **Display-app**) updated every time a change happens.

2.7.2 Roaches-client

The **Roaches-client** does not need any special user interface, although it may print the cockroaches' movements as they are generated and sent to the server.

2.7.3 Wasps-client

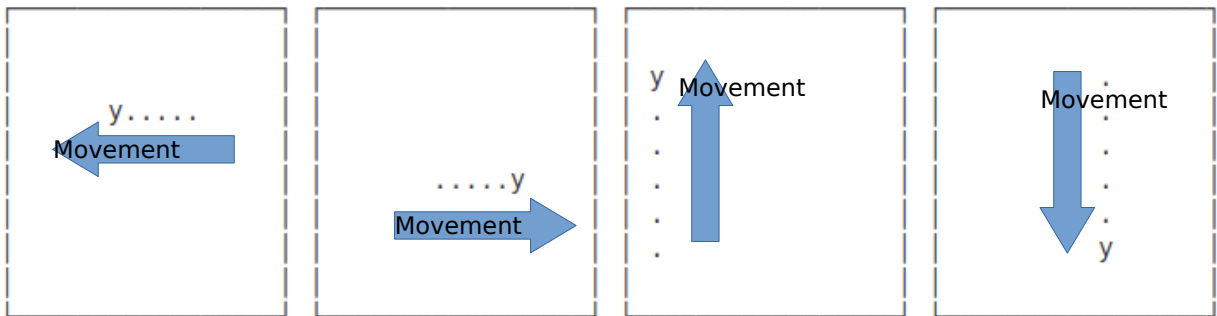
The **Wasps-client** does not need any special user interface, although it may print the wasps' movements as they are generated and sent to the server.

2.7.4 **lizardsNroachesNwasps-server** and **Lizard-client**

The **lizardsNroachesNwasps-server** and **Lizard-client** should implement a simple NCurses interface to display the field (with the lizards, cockroaches and wasps) and the scores of all the lizards.

Cockroaches should be identified by their score value (between 1 and 5) and lizards by their letter. Wasps are identified by #.

Lizards have a head (represented by its assigned letter) and a body represented by 5 dots as illustrated in the next figure. This figure also shows how lizards are drawn when moving to the left, right, up and down.



The head of the lizard is always in the direction of the movement, and when it changes direction, the head should go to the correct position, reposition itself correctly, and the body should be redrawn in the new direction.

The **lizardsNroachesNwasps-server** and **Lizard-client** should also display the game status, i.e. the scores of all lizards.

The **Display-app** should be removed from this system.

Every time a lizard, cockroach or wasp moves, the screens of these applications should be updated. Every time the score of a lizard changes the screens of these applications should be updated.

The size of the field can be defined with constants, for instance 30x30.

2.8 Cockroaches life-cycle

Cockroaches are created whenever a **Roaches-client** successfully connects and are destroyed when the corresponding **Roaches-client** disconnects. When cockroaches are created, the server places them in random positions.

When a cockroach is eaten by a Lizard it is not destroyed, it disappears 5 seconds to reappear on a random place.

Cockroaches can move to the same place as other cockroaches, can be below or on top the body of lizards, but cannot move to the head of the lizards.

Cockroaches are not affected by wasps but cannot share the place with wasps.

2.9 Wasps life-cycle

Wasps are created whenever a **Wasps-client** successfully connects and are destroyed when the corresponding **Wasps-client** disconnects. When wasps are created, the server places them in random positions.

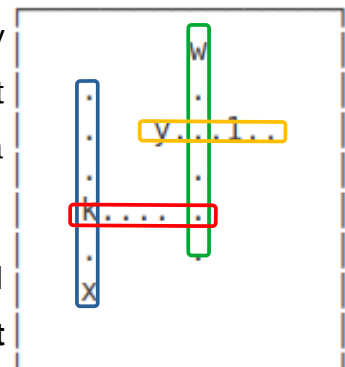
Wasps cannot share places with any other animals (wasps, cockroaches, or lizards' heads). Wasps can share place with the body of lizards.

When a wasp bumps into the head of a lizard or a lizard head bumps into a wasp, the score of the lizard decreases, but they do not change places.

2.10 Lizards life-cycle

A lizard is created whenever a **Lizard-client** successfully connects and is destroyed when the corresponding client disconnects. When a lizard is created the server places it in a random position and its score is zero.

The position of the lizard head and direction of the body should be calculated on the server when receiving **Lizard_movement** messages from the **Lizard-client**.



The head of the lizards cannot share a position with any other lizards' head, cockroaches or wasps. The body of a lizard can overlap lizards (head and body), cockroaches and wasps.

When a lizard eats a cockroach, the score of the lizards increases the value of the cockroach and the cockroach disappears.

When a wasp bumps into the head of a lizard or a lizard head bumps into a wasp, the score of the lizard decreases by 10 points, but they do not change places.

When a lizard reaches a score of fifty (50) it wins, and its body starts to be draw with *: *****y

When a lizard reaches a negative score, it loses, its body disappears and only the head is drawn.

3 Clients disconnect

There are two ways for clients to disconnect and be removed from the game:

- disconnect messages;
- Timeout (which ultimately deals with an abrupt exiting).

If a client sends a disconnect message to the server, the corresponding animals (lizards, cockroaches or wasps) are removed from the field.

After 1 minute of inactivity of a client, it should be removed from the game. If the client later interacts with the server, it should be notified that the messages are invalid.

4 Interoperability

The interface between the server and the **Wasps-client** / **Roaches-client** and the server should be supported by Protocol Buffers for message encoding.

Optionally, and as a bonus in the evaluation, students can implement one of these clients (**Wasps-client** and/or **Roaches-client**) using a different programming language (java, python or other).

5 Project Development technologies

For the communication of the various components, students should only use **ZeroMQ TCP sockets**.

Students should implement the system using standard C (ANSI or C99) **without** resorting to the following:

- ~~threads~~; (students **should** use threads)
- select;
- non-blocking communication;
- active wait.
- ZeroMQ patterns not taught/presented in the classes

6 Error treatment / Cheating

When implementing a distributed/network-based system, servers cannot guarantee that the clients lawfully abide to the defined protocol.

If the communication protocol permits it, malicious programmers can exploit the devised messages and interactions for cheating purposes.

Besides verifying all the received messages on the server to detect errors in communication, the protocol and data exchanged between clients and server should guarantee that no cheating can be performed by a malicious client that subverts the semantic and order of the messages.

Here, we are not addressing hacking concerns that could be solved using cryptography. The code must ensure that programmers with a C compiler and knowledge of the protocol cannot disrupt the game (for instance by moving other players' lizards, ...)

7 Report

Student should prepare a simple report with:

- description of the implemented functionalities;
- description of the architecture of the systems;
- description of the communication protocols (messages and interaction);
- critical description of the changes between the two versions.

Students should present the various modules with the corresponding API, message contents and interaction diagram between components.

8 Project submission

The deadline for the submission for part B of the project will be **7th January 2024 at 19h00 on FENIX**.

Before submission, students should create the project group and register at FENIX.

Students should submit a unique **zip** file containing the code for all the components. Since the complete system is composed of a server and multiple clients, each of the developed programs should be placed in different directories. One or multiple Makefiles for the compilation of the various programs should also be provided by the students.

The report (described in Section 7) should also be submitted to FENIX as a PDF file.

9 Project evaluation

The grade for this project will be given taking into consideration the following:

- Number of functionalities implemented
- Communication
- Code structure and organization

PSis 23/24 - 🦎 lizards N roaches N wasps 🐛 (Part B)

- Error validation and treatment
- Cheating robustness
- Comments
- Report

