

Systems Programming

2023/2024

Project Assignment – Part A



In the first part of the project, students will implement a simple variation of the snake game, where lizards walk in a field and eat cockroaches.

The server manages the field, while lizards and cockroaches are controlled by different programs/clients that send movement indications to the server. There are applications that displays the field in progression throughout the game.

1 lizardsNroaches

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

Each lizard score varies as follows:

- When a lizard rams another one the heath 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

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

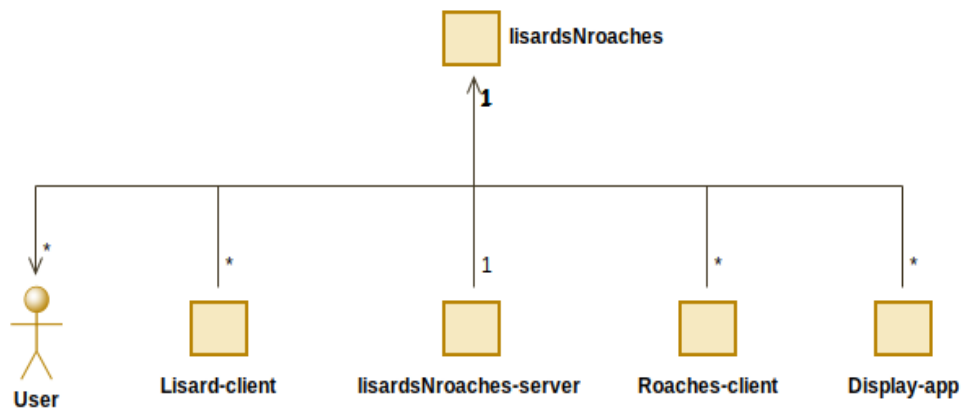
A lizard wins the game when it reaches 50 points.

2 Project Part A

In the first part of the project, students will implement the **lizardsNroaches** game using a distributed architecture, where each lizard is controlled by a user, the cockroaches are controlled by roaches bots (another type of client), and displays can show the game evolution:

PSis 23/24 - 🦎 **lizardsNroaches** 🐜 (Part A)

- **lizardsNroaches-server** is a server application that receives messages from all the clients, handles the board/files (lizards and cockroaches) and send updates to the **display-app**. This application shows the board with all the participants.
- **Lizard-client** is an application that reads the keys pressed by the user, sends them to the server, and receives the user score.
- **Roaches-client** is an application that sends random movements for a set of 10 roaches.
- **Display-app** is an application that shows the board with all the participants in the same way as the **lizardsNroaches-server**. Multiple **Display-app** can run simultaneously.



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

The **Lizard-client**, **Roaches-client** and **Display-app** 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 is waiting for messages from the clients, 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)
- **Disconnect + response** (from **Lizard-client** to the server)
- **Field_update** (from the server to the **Display-app**)

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** send a **Lizard-movement**, the server should update the board, update the scores, reply to the client and send a **Field_update** message to the **Display-app'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 messages to the **Display-app's**.

If multiple **Lizard-client's** are connected, the server will only process one **Lizard_movement** at a time and will only send the update of score to the player whose movement was processed. If a **Lizard-client** client does not send any **Lizard_movement** it will not get updates on the Lizard score.

2.2 lizardsNroaches-server

The **lizardsNroaches-server** is a C program with 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.

Cockroaches cannot occupy more than 1/3 of the field. Students should decide what happens if a **Roaches-client** tries to connect and there already are enough cockroaches in 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** or **Roaches-client**) is inserted into a list or array when the server receives a **Lizard_connect** or **Roaches_connect** messages and is removed when the server receives a **Lizard_disconnect** message.

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.

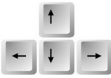
The address and port of the server should be supplied to the program as a command line argument.

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 is 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.

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 movement 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.

2.5 Display-app

The **Display-app** is a C program that interacts with the server using **ZeroMQ TCP sockets**. This program mirrors the content displayed by the server on its screen and is updated each time there is a movement by lizards or cockroaches.

2.6 User interfaces

2.6.1 Lizard-client

The **Lizard-client** should implement a simple NCurses interface to allow reading the cursor keys and print the lizard score. This application should not show the game field.

2.6.2 Roaches-client

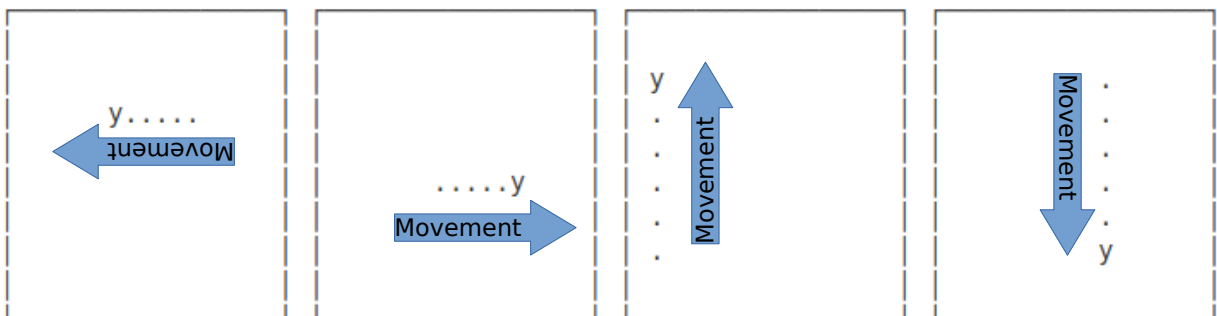
The **Roaches-client** does not need any special user interface, although it may print the cockroaches movements as they are generated and sent to the server.

2.6.3 lizardsNroaches-server and Display-app

These two application should implement a simple NCurses interface in order to display the field (with the Lizards and cockroaches) and the scores of all the lizards

Cockroaches should be identified by its score value (between 1 and 5) and lizards by its letter.

Lizards have an 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 **lizardsNroaches-server** and **Display-app** should also display the game status, i.e. the scores of all lizards.

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

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

2.7 Cockroaches life-cycle

Cockroaches are created whenever a **Roaches-client** successfully connects and are never destroyed. When cockroaches are created, the server places them in random positions.

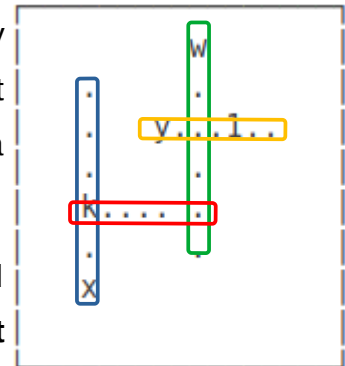
When a cockroach is eaten by a Lizard it is not destroyed, it disappears 5 seconds to reappear on a random place.

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

2.8 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 lizard's head but can overlap with the bodies of other lizards. The body of a lizard can be positioned above or below cockroaches.

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

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

3 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;
- select;
- non-blocking communication;
- active wait.

4 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 exploiy 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, ...)

5 Project submission

The deadline for the submission for part A of the project will be **10th December at 19h00 on FENIX**.

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

Students should submit 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.

6 Project evaluation

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

- Number of functionalities implemented
- Communication
- Code structure and organization
- Error validation and treatment
- Cheating robustness
- Comments