Systems Programming

2024/2025

The Project – Part A

SpcInvdrs

In the first part of the project, students will implement a simple variation of the Space Invaders game.

The server (**game-server.c**) manages outer space, where aliens move and astronauts try to shoot them. Players run simple client applications (**astronaut-client.c**) that connect to the server to allow the control of the humans. Another application (**outer-space-display.c**) displays the outer space and updates it along with the game progression.

1 SpcInvdrs

In the **SpcInvdrs** game, each user controls an astronaut who moves through outer space and tries to zap (with a laser gun) the aliens. The aliens move randomly.

Every time an astronaut fires the laser and shoots an alien, a point is awarded to the astronaut.

The game ends when all allies are destroyed, and the astronaut with more kills wins.

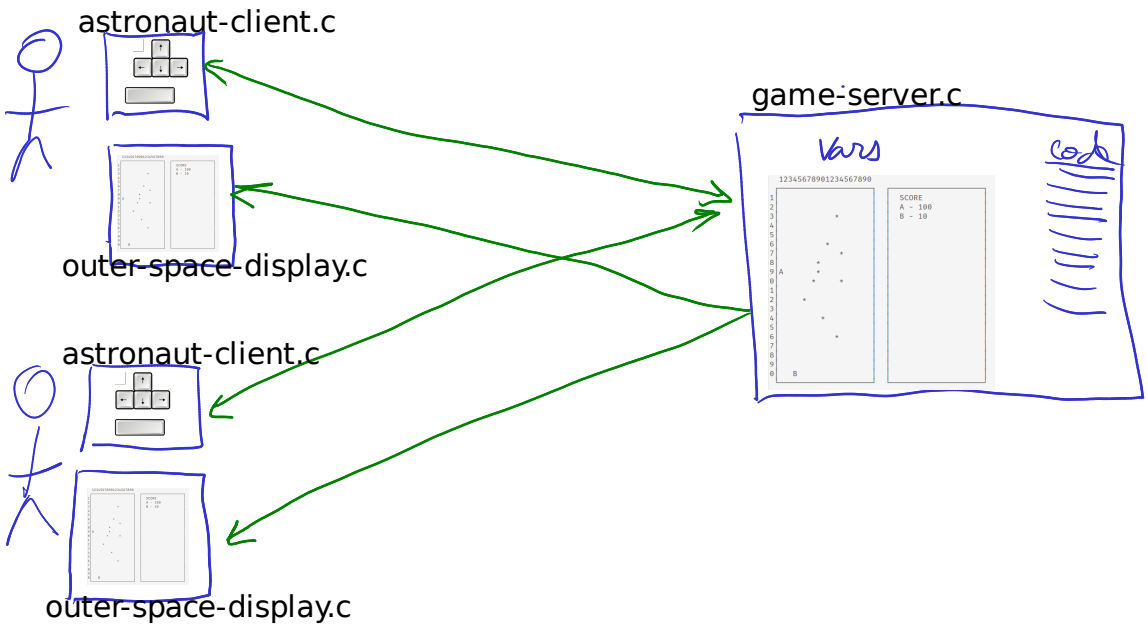
2 Project Part A

In the first part of the project, students will implement the **SpcInvdrs** game using a distributed architecture, where a user controls each astronaut, the aliens move randomly, and displays can show the game's evolution:

- **game-server.c** is a server application that receives messages from all the clients, handles outer space with all the aliens and astronauts, implements the

game rules, and sends updates to the **outer-space-display.c**. This application also shows the outer space.

- **astronaut-client.c** is an application that reads the keys pressed by the user, corresponding to the movement of one astronaut, sends them to the server, and receives the user score (how many aliens were zapped).
- **outer-space-display.c** is an application that shows the outer space with all the participants in the same way as the **game-server.c**. Multiple **outer-space-display.c** can run simultaneously.



Up to 8 users can play simultaneously (launching various **astronaut-client.c**).

The **astronaut-client.c**, and **outer-space-display.c** are independent processes/clients that interact with the **game-server.c** using **ZeroMQ TCP sockets**.

Every astronaut should be identified by a unique letter. The **game-server.c** should assign this letter when the **astronaut-client.c** first connects.

The aliens move randomly in outer space at a rate of one place per second.

2.1 Interaction

The server is waiting for messages from the **astronaut-client.c**. Depending on the message received, it changes its state and replies to the client accordingly.

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

- **Astronaut_connect + response** (from the **astronaut-client.c** to the server)
- **Astronaut_movement + response** (from the **astronaut-client.c** to the server)
- **Astronaut_zap + response** (from the **astronaut-client.c** to the server)
- **Astronaut_disconnect + response** (from **astronaut-client.c** to the server)
- **Outer_space_update** (from the server to the **outer-space-display.c**)

Every **astronaut-client.c** needs to send an **Astronaut_connect** message at startup. The server stores the relevant client and astronaut information in an internal list/array of clients and replies to the current client. Whenever the **astronaut-client.c** sends a message to the server, the server should update the outer space, update the scores, reply to the client, and send an **Outer_space_update** message to all the **outer-space-display.c**.

If multiple **astronaut-client.c** are connected, the server will process only one message at a time and will only send score updates to the player whose movement was processed. If an **astronaut-client.c** sends no message, it will not get updates on the other astronaut's scores.

2.2 game-server.c

The **game-server.c** is a C program that interacts with the other game components using ZeroMQ TCP sockets.

The maximum number of simultaneous players is eight (8). Students should decide what happens to a client that sends an **Astronaut_connect** message when 8 clients are already connected.

At the beginning of the game, 1/3 of the outer space should have aliens.

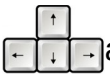
The server should store all the clients and relevant information (e.g., player position, score, …) in lists or arrays. Every **astronaut-client.c** should be inserted into a list or array when the server receives an **Astronaut_connect** message and removed when the server receives an **Astronaut_disconnect** message.

2.3 astronaut-client.c

The **astronaut-client.c** is a C program that interacts with a server using **ZeroMQ TCP sockets**. This program allows a user to control an astronaut in the outer space.

The **astronaut-client.c** should implement a simple NCurses interface to read the cursor keys and print the astronaut score. This application should not show the game field.

In order to simplify the execution of the various applications the addresses (IP address and port) of all the sockets in the project should be defined in a .h file

The astronaut is controlled using the cursor keys,  and the gun is fired by pressing the space-bar.  After reading the astronaut direction from the keyboard, this program forwards it to the server with a message **Astronaut_movement**. When firing the laser, this application send to the server a **Astronaut_zap** message.

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

Before sending any **Astronaut_movement** or **Astronaut_zap** messages, the **astronaut-client.c** should connect to the server (message **Astronaut_connect**) and

receive the assigned letter. Only after receiving this message does the client go into the loop that:

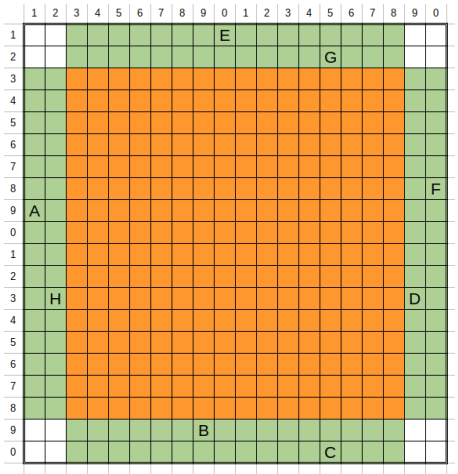
- reads a key press;
- sends the respective **Astronaut_movement** or **Astronaut_zap** message to the server;
- receives a reply with the astronaut score;

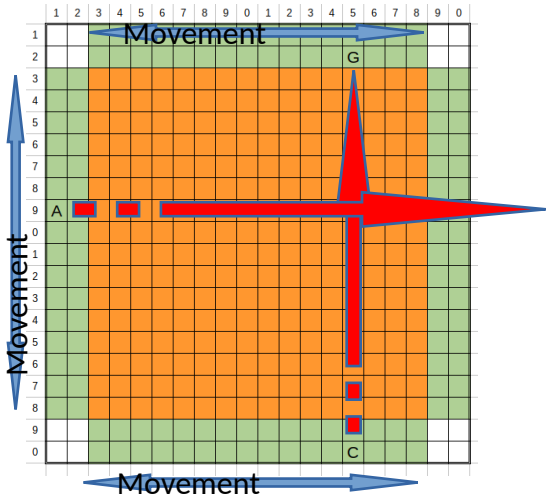
2.4 outer-space-display.c

The **outer-space-display.c** is a C program that interacts with the server using **ZeroMQ TCP sockets**. It mirrors the content displayed by the server on its screen and is updated each time an astronaut or alien moves.

2.5 Outer space organization

The outer space is a **20x20** square where the astronauts occupy the edges, and the aliens occupy the center:





The aliens move in the inner part of outer space (orange cells), and the astronauts move in the outer part of the outer space (green cells).

Astronauts either move sideways or up/down, depending on the regions of outer space where they are placed. There are 4 regions that allow the astronaut to move sideways

(corresponding to astronauts E, G, B, and C of the previous figure) and 4 regions that allow the astronauts to move up/down (astronauts A, H, D, and F). Each of these 4 regions can only be occupied by one astronaut assigned by the server when processing the **Astronaut_connect** message. The previous image shows the various combinations of astronaut placements.

Astronaut movements are limited by the region they are in:

- they either move sideways or up/down
- cannot move to another region
- cannot go to the white areas

Aliens move randomly in the orange areas at a speed of one place per second.

2.6 Astronaut zapping

The astronauts try to zap the aliens by firing their laser guns. The laser zaps (represented by red arrows) are perpendicular to the corresponding astronaut movement.

Laser zaps are so powerful that they transverse the whole outer space and affect everything on their passage:

- Aliens are destroyed, and a point is added to the astronaut.
- Other astronauts are stunned and become unmovable for 10 seconds.

Astronauts can only fire the laser at a rate of one fire every 3 seconds.

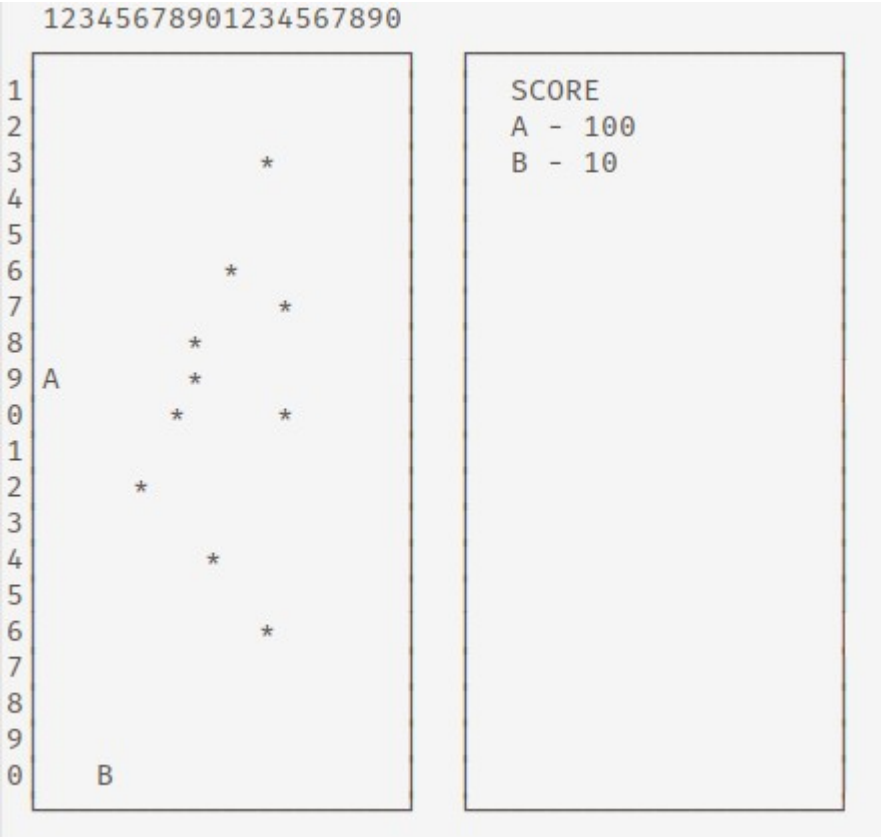
2.6.1 game-server.c and outer-space-display.c

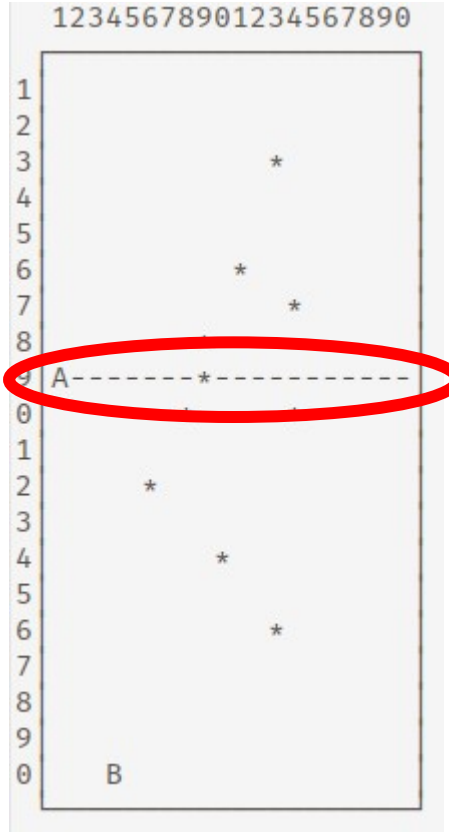
These two applications should implement a simple NCurses interface to display outer space (with astronauts, aliens, and laser rays) and the scores of all the astronauts.

Astronauts are represented by the letter assigned by the server after connecting, and aliens are represented by asterisks (*****).

The representation of a field should be done using **ncurses,** such as in the following example and following the previously presented organization. The scores should also be represented on the screen.

When an astronaut fires the laser, its ray should be drawn on the screen for 0.5 seconds. In the following example, astronaut A fired the laser.





3 Project development technologies

Students should only use **ZeroMQ TCP sockets to communicate the various components**.

Students should implement the system using the C language **WITHOUT** using the following:

- threads;
- select;
- non-blocking communication;
- active wait;
- signals.

4 Error treatment / Cheating

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

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

In addition to verifying all the messages received on the server to detect communication errors, the protocol and data exchanged between clients and the server should guarantee that a malicious client cannot cheat by subverting the semantics 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 submitting part A of the project will be **11th December 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 consists of a server and multiple clients, each developed program should be placed in a different directory.

The students should also provide one or multiple Makefiles to compile the various programs.

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