Systems Programming

2022/2023

Project Assignment – Part A

ChaseThem

In the first part of the project, students will implement a simple version of a Chase game, where balls move in a field, trying to chase other balls and pick prizes.

The server manages the field, and all balls are controlled by different programs/clients that send movement indications to the server. There are also Bots that move randomly on the fields.

1 ChaseThem Rules

In the ChaseThem game, users control a ball (represented by a letter) that moves in a field. In those fields, bots (represented by *****) move randomly. Prizes (numbered from 1 to 5) are available to be eaten.

Each ball has a health value that varies as follows:

when a ball rams into another one, there is an exchange of health:

- the ball that moved/rammed onto another one increases its health by one point;
- the other ball (the one that was rammed) decreases its health by one point;

- when a ball rams into a bot, nothing happens;
- when a bot rams into a ball, the ball's health decreases by one value;
- when a ball eats a prize, its health increases by the value of the prize.

A ball health starts at 10 and is never higher than 10 (even if eating a lot of prizes). When the ball's health reaches zero, the ball is kicked out of the game and the client terminates.

2 Project Part A

In the first part of the project, students will implement the ChaseThem game using a distributed client-server architecture, where each ball is controlled by a user. An independent process/program reads the keys pressed by the user, sends to the server the ball movements, receives the field changes, and updates the screen.

Several users can play simultaneously, and multiple bots should also be moving randomly in the field.

A new prize is placed on the board every 5 seconds.

The balls and bots controllers/clients should be independent processes/clients that interact with the server using **Unix datagram sockets**.

Every ball should be identified by a unique letter. This letter can be assigned by the server or selected by the user left as a programmer decision.

2.1 Interaction

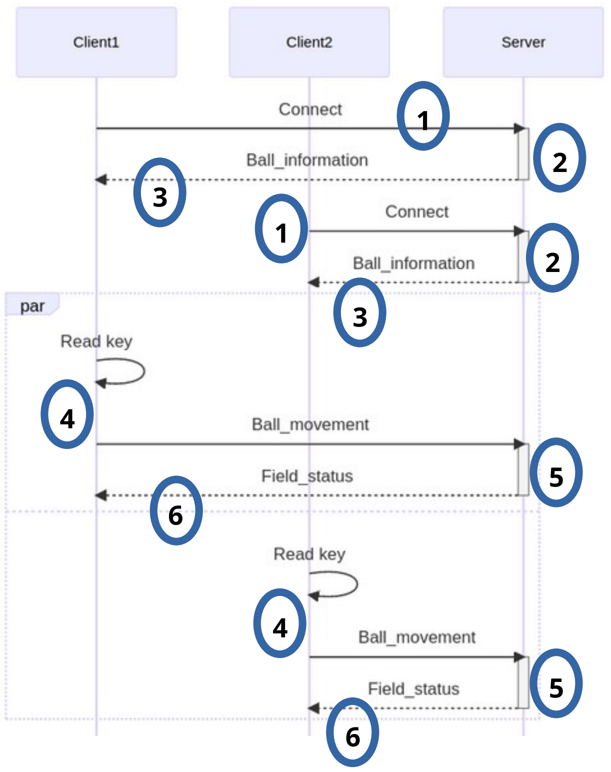
The essential interaction between the ball clients and the server is presented in the following figure.

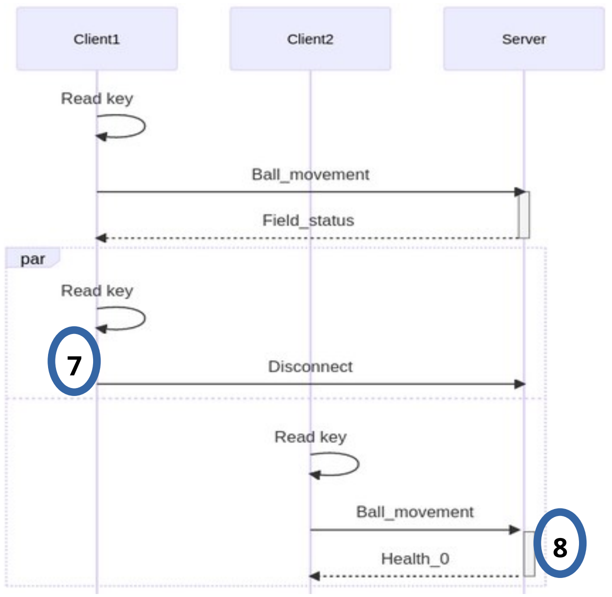
The server is waiting for messages from the clients, depending on the received message changes its state, and replies to the client accordingly.

The basic messages exchanged between the server and ball clients are:

- **Connect** (from the client to the server)
- **Ball_information** (from the server to the client)

- **Ball_movement** (from the client to the server)
- **Field_status** (from the server to the client)
- **Health_0** (from the server to the client)
- **Disconnect** (from the client to server)





Every ball client needs to send a **Connect** message (**step 1**) at startup. The server stores the relevant client and ball information in a list (**step 2**) and replies with a **Ball_information** message (**step 3**). After receiving the **Ball_information** message from the server, the user can now start controlling its player.

Every time the user moves its ball on the client (**step 4**), a **Ball_movement** message is sent to the server. The server updates the field where various balls and bots may be playing (**step 5**) and sends the new status of the field in the **Field_status** message to the player that just moved (**step 6**).

If two users/clients are connected, the server will only process one request (**step 5**) at a time and only send the update of the field to the player whose movement is being processed. After processing a ball movement, the server will read a new message and process it.

If a user/client does not send any **Ball_movement** message to the server, its screen will not be updated even if other users/clients are moving and updating its screens accordingly.

There are two ways for a player to be removed from the game:

- by initiative of the user – in this case, after reading **Q** from the keyboard (**step 7**), the client will send a **Disconnect** message to the server and then exits.
- when its health reaches zero – after receiving a **Ball_movement** message the server verifies the health level of that ball and, if it is zero replies with a **Health_0** message. When the client receives the **Health_0 message** it should exit.

When a client/ball is removed from the game (**Disconnect** or **Health_0**), the server should update accordingly its information in the array/list that stores all client information and ignore any other message from such client.

2.2 ChaseThem Server

The **ChaseThem server** is a C program with a **Unix domain datagram socket** to receive messages from various clients wanting to play the ChaseThem game.

The maximum number of simultaneous players is ten (10). Students should decide what happens to a client that sends a **Connect** message when 10 clients are already connected.

The server should store a list/array of all the clients with all the relevant information (e.g., player position, health, ...). A client is inserted into this list when the server receives a **Connect** message and is removed when the server receives a **Disconnect** message or sends a **Health_0** message.

When the server receives a **Send_movement** message, it should calculate and update the corresponding player position and global state of the field (prizes and other players health).

The maximum number of simultaneous bots is ten (10).

2.3 ChaseThem ball Client

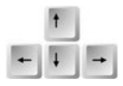
The **ChaseThem ball client** is a C program that interacts with a server using **Unix domain datagram sockets**. This program allows a user to control a ball on the game field.

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

This program reads key presses from the keyboard and forwards them to the server (if corresponding to the cursor keys) to move the user ball.

After sending the **Connect** message, the client will wait for **Ball_information** message. Only after receiving this message the client goes into the loop that:

- reads a key press;
- sends a **Ball_movement** message to the server;
- receives a reply (**Field_status** or **Health_0** messages);
- updates the client display with the new state of the Field.

The ball is controlled using the cursor keys: 

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

The client should terminate (exit) after the user presses the **Q** key or when receiving a **Health_0** message.

2.4 Bot Client

The **ChaseThem bot client** is a C program similar to the **ChaseThem ball client** that also interacts with the server using a **Unix domain datagram socket** and controls all the bots in the game.

The address and port of the server should be supplied to the program as a command line argument, along with the number of bots (up to 10).

This client is responsible for randomly deciding the movement of each bot and sending this information to the server using suitable messages defined by the students.

Every 3 seconds, this program decides in what direction the bots will move and sends a suitable message to the server. After the server updates the position of all the bots (and the health of the balls), it is not necessary to reply to this client with the status of the field.

2.5 Prizes

During the game, there should exist between zero and ten prizes; each prize has a value between 1 and 5.

When the game starts, 5 prizes are placed randomly in the field.

Every time a ball moves into a prize, the ball's health is increased by the value of the prize, and the prize disappears.

Every 5 seconds a new prize should be put into the field.

Students should decide how to place a new prize every 5 seconds and how to guarantee that there are at most ten prizes in the field.

2.6 Ball/bot movements

The ball and bot positions should be calculated on the server after receiving messages from the **ChaseThem ball clients** or **ChaseThem bot client**.

Balls and bots should:

- not share the same space (when moving or being randomly placed);
- not bounce if the ball/bot goes against a wall (it should count as a movement, but stay at the same place);
- remain in the same place when a ball/bot rams into another ball/bot.

When a ball eats a prize, the prize will disappear, and the ball should go into its place. Bots and prizes do not interact; when a bot rams into a prize it constitutes a movement, but it does not eat it and should both remain in the same place.

2.7 User interfaces

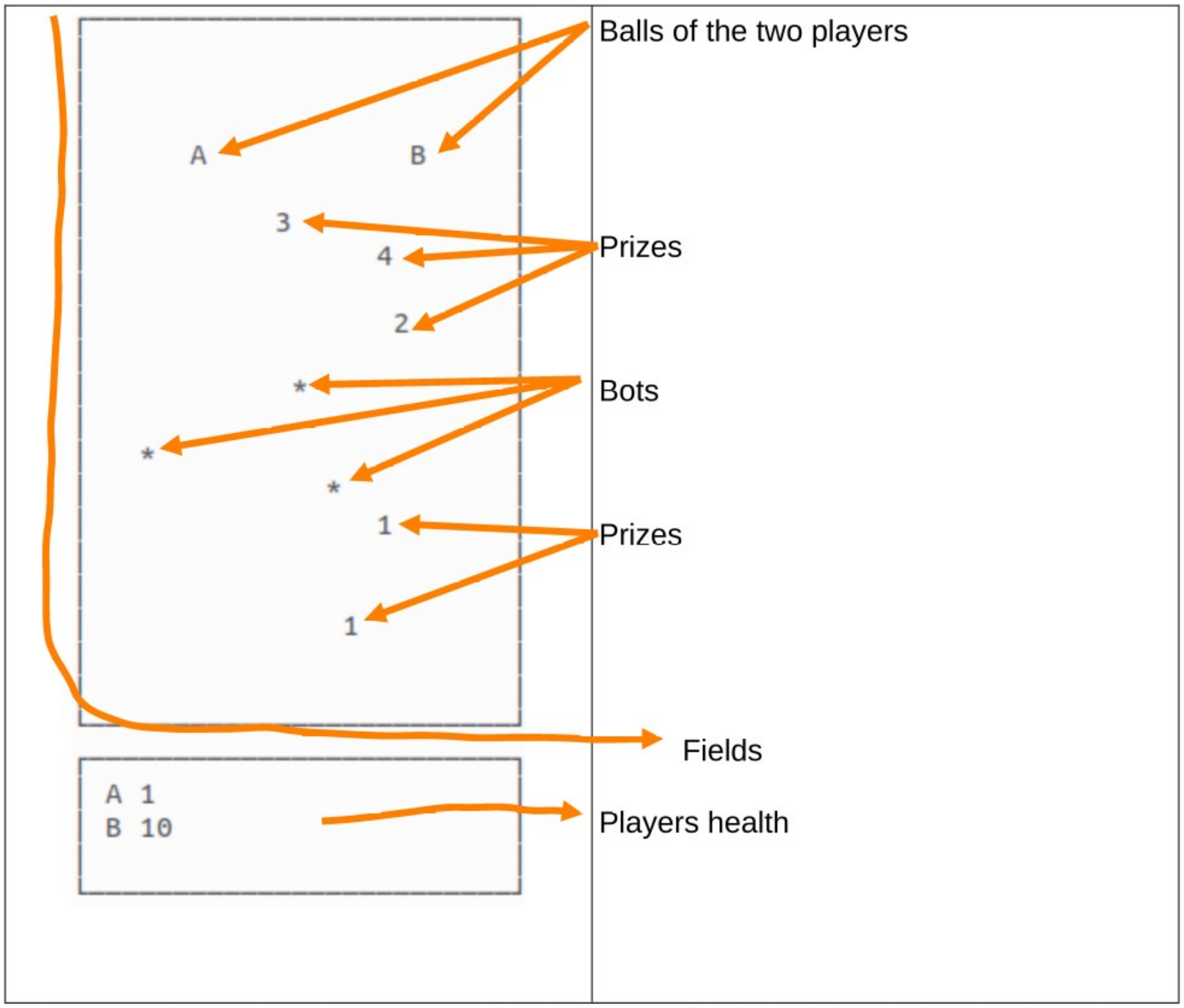
The **ChaseThem ball client** will use **ncurses** to read the keyboard and draw the balls, bots and prizes on the screen. The client only updates the screen when it receives from the server a **Field_status** message.

The **ChaseThem server** should also display the game status, but update the screen as messages from clients (ball and bot) are received and processed.

Students are provided with the sample code of the User interface using **ncurses**, that implements functions and data structures to:

- store ball information;
- initialize, draw, and move balls;
- draw the field on the screen;
- read the keyboard;
- print warnings and messages/balls health.

The client interface e should follow generic structure:



As the server receives messages it should update the location of the various balls, prizes, bots and players' health, by deleting the characters on the old positions and writing the corresponding symbols in the new positions.

The size of the field can be defined as constants in a **.h** file to be included by the clients and server (as in the provided example code).

3 Project Development technologies

For the communication of the clients with the server, or others, students should only use Unix domain datagram 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 communication protocol allows it, malicious clients can use the developed messages and interactions to cheat.

Besides the verification of 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.

5 Project submission

The deadline for the submission for part A of the project will be 1**6th December at 19h00 on FENIX**.

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

Students should submit a **zip** file containing the code for both games. Since the complete system composed 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