OPERATING SYSTEM PROJECT

VIDEOGAME

Implemented a distributed videogame.

- Server side:

The server operates in both TCP and UDP connection.

- TCP part:

1. registering a new client when it comes online;
2. deregistering a client when it goes offline;
3. sending the map, when the client requests it.

- UDP part:

1. the server receives periodic updates from the client in the form of: <translational acceleration, rotational accelerations>;
2. each “epoch”, it integrates the messages from the clients and sends back a state update;
3. the server sends to each connected client the position of the agents around him.

- Client side:

The client does the following:

1. connects to the server (TCP);
2. requests the map and gets an ID from the server (TCP);
3. receives updates on the state from the server.

Periodically:

1. updates the viewer;
2. reads either keyboard or joystick;
3. sends the <UDP> packet of the control to the server.

HOW IT WORKS

- Server side:

At first, the server takes map elevation and texture paths from terminal, loads the images, configures TCP socket, provides to signal handling and initializes the world with the images loaded before.
The server main program continues with the initialization of TCP connection through a port number defined in *common.h* launching a thread that will be joined by the main program.
The function run by this thread (namely *thread_server_tcp*) accepts every client connection and launches other two threads, *client_thread_handler* and *udp_handler*, that will manage respectively clients’ requests (id, map elevation, map texture) and clients’ updates.
Every time that the server accept a client connection, it adds the client to a specific data structure, *ClientList*, that contains all users connected, each one as a *ClientListElement*.
Each user has a unique id, correspondent to his socket TCP descriptor.
Moreover, each user has its vehicle, received by the user itself through TCP communication.
In *client_thread_handler*, the server waits for a *GetId* request from the client and answers with a *GetId* packet containing the id assigned to that client.
The second step consists in the receiving of *GetTexture* and *GetElevation* packets, that will be followed by an answer of type *PostTexture* and *PostElevation*.
The server answer with a *PostTexture* packet that contains that texture.

In *udp_handler*, the server configures the UDP socket, with a port number defined in *common.h.*
After that, the server receives a *VehicleUpdatePacket* from the client and updates the user’s data structure contained in *ClientList.*
Moreover, the server sends a *WorldUpdatePacket* to every client in users list to update position, translational and rotational force of every user.
If server is closed, it sends a *PostDisconnect* packet to every client connected in order to advise them of its disconnection.
When a user comes off-line, the server deletes its vehicle from the world and deallocates its data.
When the server is closed, it frees used resources, closes TCP socket, destroys the world and terminates.

- Client side:

At the beginning, the client takes IP address and vehicle texture path from terminal, loads this image, provides to signal handling and configures TCP connection.
After that, it connects to server and sends *GetId*, *GetElevation*, *GetTexture* requests, to receive id, vehicle texture, map elevation and map texture of the world, and a *PostTexture* packet containing its vehicle texture.
After having initialized the world and added its vehicle to it, the client main program configures socket UDP and creates two threads (*sender_udp* and *receiver_udp*).
At the end of the main program, the game is opened in a new window.
The *sender_udp* thread provides to send *VehicleUpdatePacket* to communicate vehicle’s movements to server.
The *receive_udp* thread, instead, receives *WorldUpdatePacket* to update the client’s world:
- if a vehicle isn’t in the world, the client launches a *get* thread to add the new user in the world with its vehicle;
- else if a vehicle exists, the client updates its position in the world;
- at the end, the client removes disconnected clients with their vehicles from the world.

If a *PostDisconnect* packet is received, the client releases used resources and it terminates.
When the client is closed, the used resources are released, TCP and UDP socket are closed and it terminates.

HOW TO RUN

First of all, the game has to be compiled: digit on terminal *make* (or *make -j)* .
To run the videogame digit from terminal:

- Server side:

./so_game_server <map_path/map_elevation_filename.pgm> <map_path/map_texture_filename.ppm> (without “<>”)

- Client side:

./so_game_client <ip address> <vehicle_texture_path/vehicle_texture_filename.ppm> (without “<>”)

Example:
./so_game_server images/maze.pgm images/maze.ppm
./so_game_client 127.0.0.1 images/arrow-right.ppm

Project implemented by:
Marco Costa
Andrea Luca Antonini