TECHNICAL GUIDE R-TYPE

Doc :	V1.0	**Date :**	10/11/2020
Description :	Learn how's R-Type working and how to use it		
Compatibility :	Linux & Windows		
Key Words :	SFML, VideoGame, Online		
What's :	PC VideoGame mo		
Who's :	Devs		
Who's :	By Any Dev		
Writers :	Vivien Bouvier, Rayann Folleas, Hadrien Leonard & Adrien Auge	**Techs :**	Vivien Bouvier / Rayann Folleas / Hadrien Leonard / Adrien Auge
Type of document :	Documentation RFC Compliant	**Functionals :**	Vivien Bouvier / Rayann Folleas / Hadrien Leonard / Adrien Auge
Goes over older :	None		

<u>Summary</u>

1 DESCRIPTION AND PREREQUISITES..2

1.1 When to use this Guide?..2

1.2 What do I need to use this guide?..2

2 HOW DO I … 2

2.1 Launch the server? 2

2.2 Launch and play the game? 2

2.3 [Where] Find the source code & how is it working? 2

2.4 Identify the game objects..4

2.5 Modify the app code?..5

2.6 Test my new code? 6

2.7 Annex 6

1 Description and prerequisites

1.1 When to use this Guide?

You have just accessed the R-Type online videogame application created by Rayann Folleas, Hadrien Leonard, Adrien Auge & Vivien Bouvier and you want a technical guide to understand or modify the code? You must read this document.

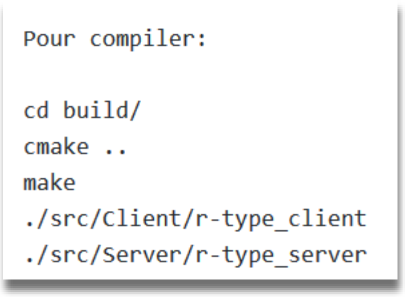
1.2 What do I need to use this guide?

A Computer (wireless component)
Linux & SFML installed
A good knowledge of C++ & SFML facilities. We're also using Boost::Lib for multithreading the server

2 How do i …

2.1 Launch the server?

First, you need to know how to launch a cmake. Cmake list => gives you a Makefile. And you've two Makefiles to launch, one for the client(x number of players), one for the server. Here's an example:



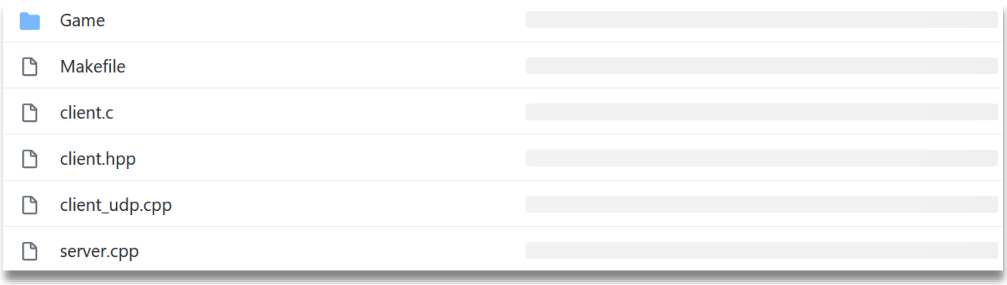
```
Pour compiler:

cd build/
cmake ..
make
./src/Client/r-type_client
./src/Server/r-type_server
```

2.2 Launch and play the game?

Once the server is working, you'll just have to launch any number of clients you like. Just remember it's 4 players in game max. You'll also need to be connected on the same Wi-Fi as the server.
Here's the keys you must use to play out: ZQSD – move | Spacebar – shoot | Esc – pause/leave

2.3 [Where] Find the source code & how is it working?

As you may have noticed, there's two parts for the code, server and client. First you need to know what you're searching for: if it's the server, check the server file : ./Server/



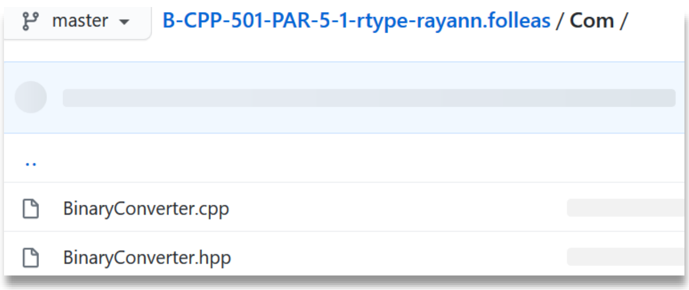
```
Game
Makefile
client.c
client.hpp
client_udp.cpp
server.cpp
```

Here comes a little explication: the server and his part for communication is contained in the server.cpp file, and the historical tests for the clients are in client_udp.cpp.

The second part of the code is what relies server and client: the communication.
Here's how work the binary protocol:

Server(structure) => encoding in binary => (ex: 01100010.10101010.10101101.01010111.00000001.01000001) as you may see, it's coded in 6 bytes for the server/client communication. => client => decoding back to the structure => first byte = first variable in the structure.
In fact, we encode a structure by bytes and decode bytes.

You'll find the communications files: ./Com/

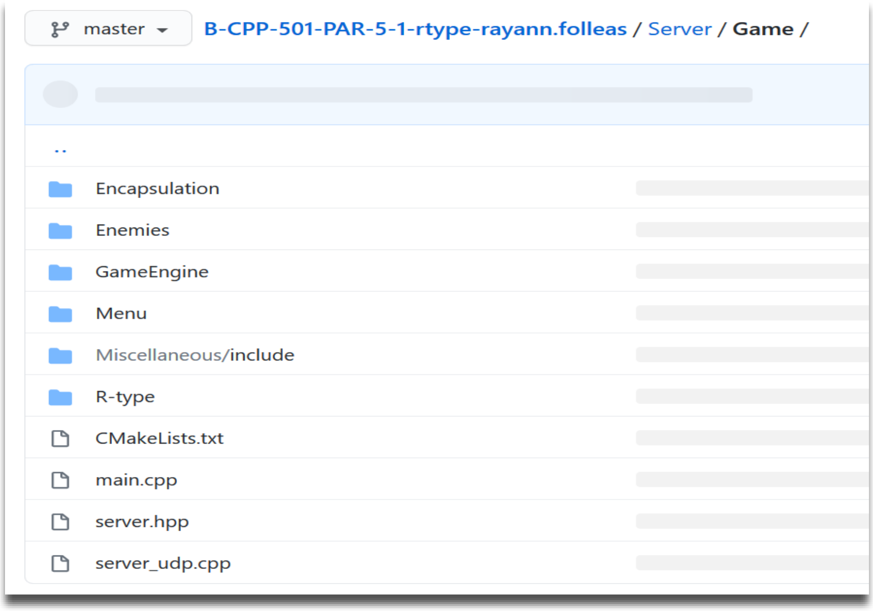

master B-CPP-501-PAR-5-1-rtype-rayann.folleas / Com /

..
BinaryConverter.cpp
BinaryConverter.hpp

This part has been fully tested, it mustn't be modified

Last part of the code is the game itself. Our protocol figures out a strong server and a weak client to avoid any forms of cheating. And so is the game working, the client only displays the game and send player's inputs to the server.

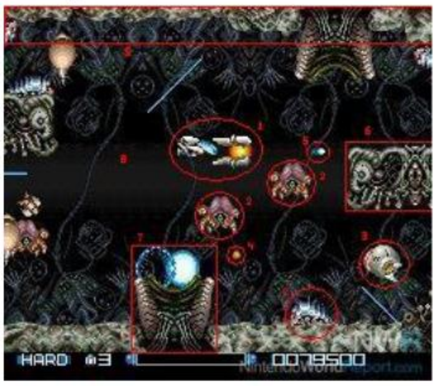
You'll find out the code right here: ./Server/Game/ (as the server contains the game)


master B-CPP-501-PAR-5-1-rtype-rayann.folleas / Server / Game /

..
Encapsulation
Enemies
GameEngine
Menu
Miscellaneous/include
R-type
CMakeLists.txt
main.cpp
server.hpp
server_udp.cpp

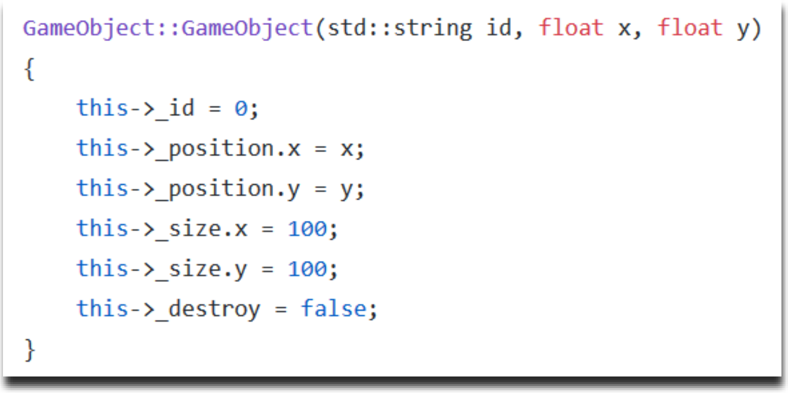
2.4 Identify the game objects

When you have launched the game, you'll be able to see all these elements in the game. Here an example screen of the game:



- 1: Player
- 2: Monster
- 3: Monster (that spawns a powerup upon death)
- 4: Enemy missile
- 5: Player missile
- 6: Stage obstacles
- 7: Destroyable tile
- 8: Background (starfield)

Each object on the screen correspond to a game object made in the Server/Game/GameEngine/src/GameObject.cpp file:



```
GameObject::GameObject(std::string id, float x, float y)
{
    this->_id = 0;
    this->_position.x = x;
    this->_position.y = y;
    this->_size.x = 100;
    this->_size.y = 100;
    this->_destroy = false;
}
```

Note: If you need to understand to how to play to R-Type game, please refer to user guide or check it on the internet!

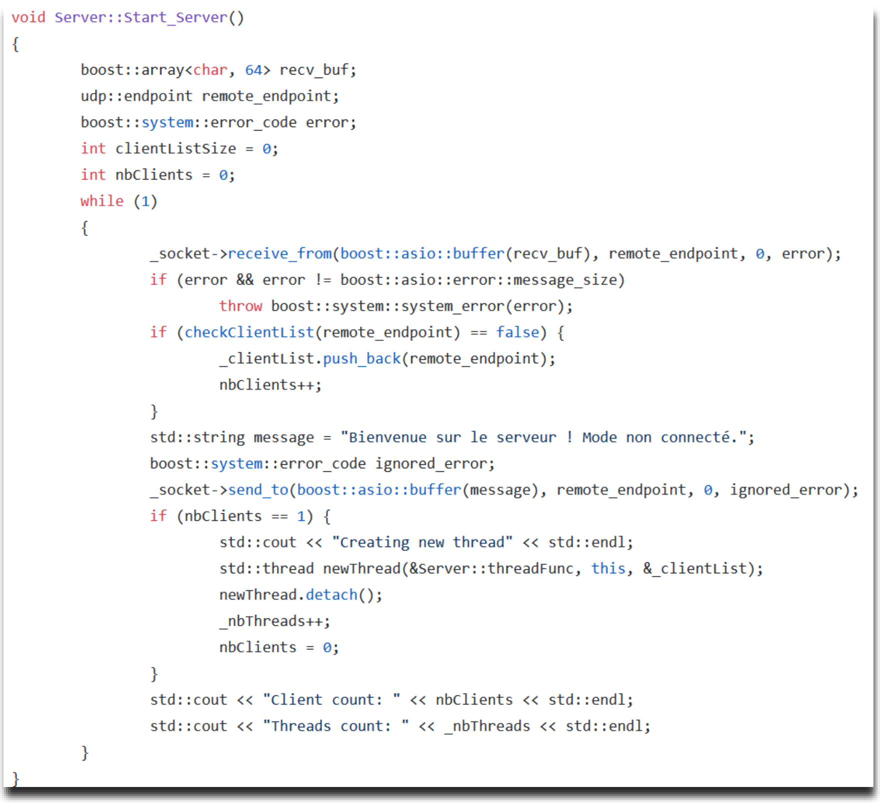
2.5 Modify the app code?

You can use any code editor. A solid knowledge of C++ is essential to be able to modify the application code. If you need to change a specific part of the code, consider updating the rest of the code accordingly.

If you change a visual aspect of the application, you must check the display of your new components. Note that all the code is broken down into independent components which are reusable in the event of modification of another part of the code. Thanks to encapsulation!

So here we'll just show you the start of any part, like the starting point of any parts:

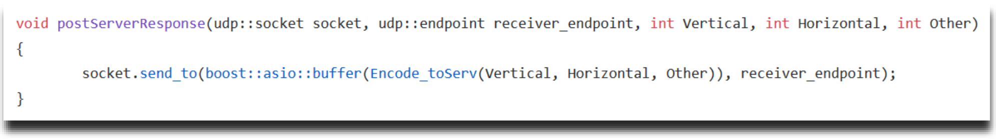
1. ***Server:***



```cpp
void Server::Start_Server()
{
        boost::array<char, 64> recv_buf;
        udp::endpoint remote_endpoint;
        boost::system::error_code error;
        int clientListSize = 0;
        int nbClients = 0;
        while (1)
        {
                _socket->receive_from(boost::asio::buffer(recv_buf), remote_endpoint, 0, error);
                if (error && error != boost::asio::error::message_size)
                        throw boost::system::system_error(error);
                if (checkClientList(remote_endpoint) == false) {
                        _clientList.push_back(remote_endpoint);
                        nbClients++;
                }
                std::string message = "Bienvenue sur le serveur ! Mode non connecté.";
                boost::system::error_code ignored_error;
                _socket->send_to(boost::asio::buffer(message), remote_endpoint, 0, ignored_error);
                if (nbClients == 1) {
                        std::cout << "Creating new thread" << std::endl;
                        std::thread newThread(&Server::threadFunc, this, &_clientList);
                        newThread.detach();
                        _nbThreads++;
                        nbClients = 0;
                }
                std::cout << "Client count: " << nbClients << std::endl;
                std::cout << "Threads count: " << _nbThreads << std::endl;
        }
}
```

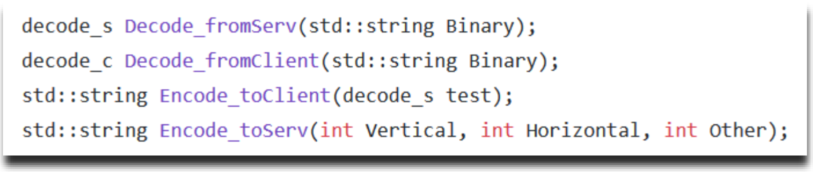
Here's the starting point of the serv, you must follow the functions.

2. ***Communication:***



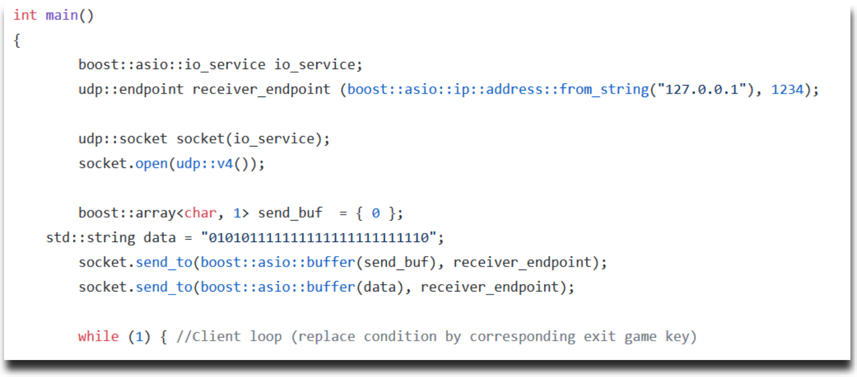
```cpp
void postServerResponse(udp::socket socket, udp::endpoint receiver_endpoint, int Vertical, int Horizontal, int Other)
{
        socket.send_to(boost::asio::buffer(Encode_toServ(Vertical, Horizontal, Other)), receiver_endpoint);
}
```

That's where the communication is called, but not the start of it, refer to Com/BinaryConverter.cpp to get the source code of the communication binary protocol:



```cpp
decode_s Decode_fromServ(std::string Binary);
decode_c Decode_fromClient(std::string Binary);
std::string Encode_toClient(decode_s test);
std::string Encode_toServ(int Vertical, int Horizontal, int Other);
```

3. Game



```
int main()
{
        boost::asio::io_service io_service;
        udp::endpoint receiver_endpoint (boost::asio::ip::address::from_string("127.0.0.1"), 1234);

        udp::socket socket(io_service);
        socket.open(udp::v4());

        boost::array<char, 1> send_buf  = { 0 };
    std::string data = "010101111111111111111111110";
        socket.send_to(boost::asio::buffer(send_buf), receiver_endpoint);
        socket.send_to(boost::asio::buffer(data), receiver_endpoint);

        while (1) { //Client loop (replace condition by corresponding exit game key)
```

Obviously, what you're watching here it's the start of the client, remember that the game is in the server, and then this is basically a streaming like working function.

Changing game code calls obligatorily to change the server code.

2.6 Test my new code?

Please refer to the 2.1 & 2.2 parts for how to view the application.

Note that some changes may require relaunching the application. You just have to repeat the procedure described upper. And don't forget some parts are essentials to the project to work, you'll have to change them in any time.

2.7 Annex

Repository's tree summary:

➔ Com/BinaryConverter(Cpp)

➔ Server > Server.cpp

> Game -> GameCode(Cpp)

➔ Client > ClientCode(Cpp)

You can find this repo on Github (you need rights to access the repository)

https://github.com/EpitechIT2020/B-CPP-501-PAR-5-1-rtype-rayann.folleas