Advanced C++ Project

R-Type

David Giron thor@epitech.net
Maxime Montinet zaz@epitech.net

Abstract: For your second project of the ***Advanced C++*** *knowledge unit,* ***R-Type*** *will introduce you to networked video game developement.*

You will have to implement a multi-threaded server and a graphical client. Also, your game has to be fun to play!

Table des matières

Chapitre I

Introduction

For the sad ignorant people who don't know this best-seller of video games, that made us lose hours and hours of our childhood, creaming many kinds of aliens each bigger, uglier and trickier than the others, here is a web site useful to make up for lost time : `http://www.hardcoregaming101.net/rtype/rtype.htm`.

As you now understand, you have to make your own version of `R-Type`, great isn't it ?

The purpose of this project is to create a one to four players game, based on a client / server architecture. And I mean it ! It `MUST` be a client / server achitecture, no peer-to-peer allowed.

> ⚠ You should take a look at this video. It's the video from the conference "Introduction to the R-Type project" made by The Game Dev Lab. If you were not there, then watch it.

Chapitre II

Server

- The server `MUST` be multi-threaded. As a consequence, you `MUST` design an abstraction to the Windows/UNIX threads and socket APIs.
- The server `MUST` be able to handle more than one game at a time, it `MUST` be able to handle multiple games in a row, and it `MUST` be the referee of all games it manages.
- Your abstractions' quality will be strongly evaluated during the final defense, so pay a LOT of attention to them.
- I strongly advise you go to the threads abstraction practical session...

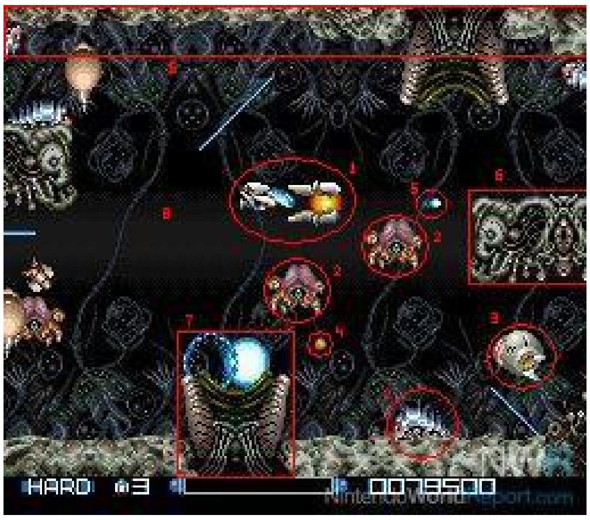
Chapitre III

Client

The client is the display terminal of the game.

- It MUST contain anything necessary to display the action and handle inputs.
- You MUST use the SFML library for this.

Here is a description of the official R-Type screen :



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

FIGURE III.1 – Annotated visual of the `R-Type` game screen

Chapitre IV

Requirements

IV.1 Protocol

- You `MUST` design a binary protocol for the client/server communication.
- You `MUST` use UDP protocol for communications between the server and the clients. A second connection using TCP could be tolerated but you `MUST` provide a quite strong justification. In any event, ALL in game communications `MUST` use UDP.
- You `MUST` document your protocol. The documentation `MUST` be an RFC. RFC format is described in the RFC 2223.
- Your RFC `MUST` look like an official RFC
- You `MUST` make the RFC in ASCII format. Postscript format is optional
- You `MUST` respect standard RFC key words given in the RFC 2119.
- You `MAY` choose to write the documentation in english or french. You are strongly invited to write the documentation in english.
- If you choose the french version, the key words of the RFC 2119 `MUST` be replaced by : "DOIT", "NE DOIT PAS", "OBLIGATOIRE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDE", "PEUT", "OPTIONNEL".
- You can find a good formated RFC here : RFC 5734. We recommand you read it, not for its contents, but as a model for your RFC.

IV.2 Language

- Only C++ is allowed. Neither C nor "C+" will not be tolerated (You will lose a LOT of points !).
- Pay attention to your use of "const", references, etc . . .

IV.3 Libraries

- You `MUST` use the SFML multimedia library on the client side.
- You `MAY` use Boost and/or Qt libraries for the client side ONLY. Please note that every graphic `MUST` be done using the SFML library. As a consequence, any graphical resource from Qt is FORBIDDEN.
- You are NOT allowed to use ANY library on the server side! This include Qt and Boost.

IV.4 General

- The client `MUST` display a slow horizontal scrolling background figuring in space, with stars, planets, anything... This is the "starfield".
- The scrolling of the starfield must NOT be synced with the CPU, instead, you `MUST` use timers.
- Players `MUST` be movable by arrow keys
- The server `MUST` be multi-threaded
- If a client crashes for any reason, the server `MUST` continue to work and `MUST` notify other clients in the same game that a client crashed.
- As a consequence, you `MUST` write an abstraction for the Windows and UNIX threads and sockets APIs.
- `R-Type` sprites are freely available on the net, but a set of sprites is available with this subject for lazy students.
- The four players in a game `MUST` be blue, red, yellow and green respectively
- There `MUST` be Bydos slaves in your game.
  - Each kind of monster `MUST` be a dynamic library that your server can add without being restarted.
  - You `MUST` write your own API to deal with those libraries.
  - Monsters `MUST` be able to spawn at the right side of the screen randomly.
  - The server `MUST` notify each client when a monster spawns, is destroyed, fires, kills a player and so on...
- This is the minimum, you `MUST` add to your game anything you feel will get it as closer to the original one.

- The mandatory SCM for the `R-Type` project is GIT. Yours repositories will be created as soon as the inscriptions are closed. See your intranet for details.

Chapitre V

Timeline

V.1 Conception follow-up

- You `MUST` provide an RFC for your protocol.
- You `MUST` provide an UML class diagram for both client AND server. This diagram `MAY` be splitted in two sub-diagrams. It `MUST` be printed and readable !
- You `MUST` provide an UML sequence diagram for both client AND server. This diagram `MAY` be splitted in two sub-diagrams. It `MUST` be printed and readable !

V.2 Implementation follow-up

- You `MUST` have a fully functional server.
- It `MUST` be possible to connect a random number of graphic terminals (up to 4 per game instance)
- It `MUST` be possible to move players using a graphic terminal. The movement `MUST` be dispatched to all the game clients.
- Collisions between two players `MUST` be detected. This feature is only mandatory for this follow-up, as a proof of concept. An `R-Type` without friendly collisions will be accepted at the final defense.

V.3 Final defense

- You `MUST` have both client and server, fully functional, in accordance to the above Requirements.
- You `MUST` respect your conception, or you WILL lose points.

- You MUST bring all your documentation, including UML diagrams AND protocol RFC.
- The KOALAs and PANDAs MUST enjoy playing your game!

Chapitre VI

Some advice

- SFML stands for Simple and Fast Multimedia Library. As its name says, this library is quite simple. As a consequence, don't panic if you have never used it. Read tutorials and the official documentation, that can be found here.
- It's a good idea to split your display in several rectangles to simplify the sprites scrolling. Watch Figure VI.1 to understand.
- Following the previous advice, a game stage can be compared to a MIDI partition. As the MIDI partition describes which instrument must play what and when, the stage 'partition' describes which sprite must be displayed, where and when.

- As I promised, `R-Type` is a very fun but difficult projet, so be fully involved and work hard. It will be impossible otherwise.
- Send your UML diagrams to Francois Carruba for feedback !
- Koalas have set up an irc chan available to the students. The server is 'irc.epitech.net', chan '#koala'. You're welcome anytime just to say hello or ask any question.
- As R-Type is a video game, I strongly recommend you to go to the Game Dev Lab (Voltaire underground) to ask technical questions about video games. They will be happy to help you and the lab directors granted their permission. Be polite, they will help you during their free time, so don't abuse of their kindness.
- As usual, you can find useful information here.

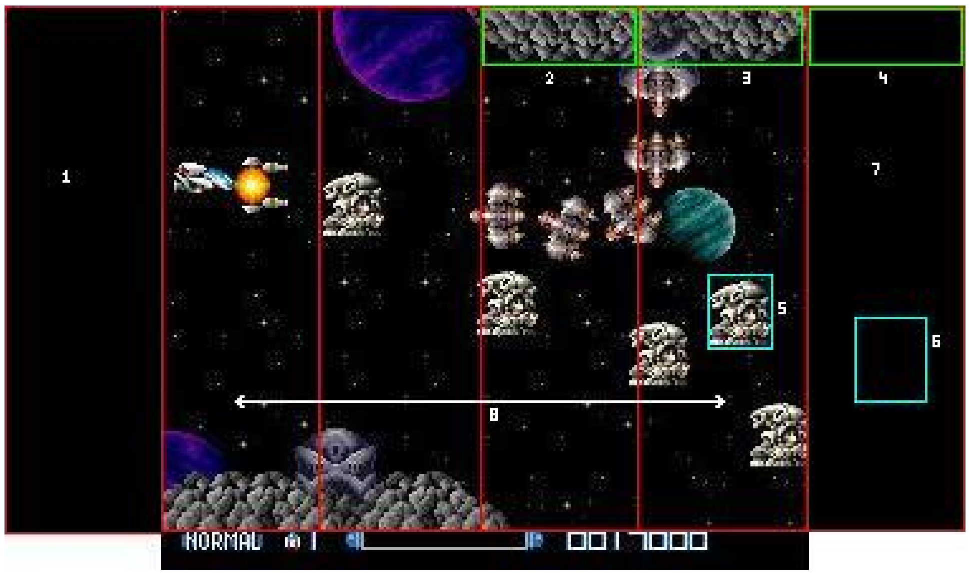


FIGURE VI.1 – "Tiled" display area for the `R-Type` game client

- 1 → Unload area. When an entire sprite is in that area, it can be unloaded.
- 2 → Ceiling sprite 1
- 3 → Ceiling sprite 2
- 4 → Area where the ceiling sprite 3 will be loaded
- 5 → Monster sprite. The cyan rectangle figures the sprite's hitbox
- 6 → Area where the next monster sprite will be loaded
- 7 → Loading area, when all sprites are loaded, ready to be displayed
- 8 → Display area

Chapitre VII

General instructions

You are (more or less) free to do the implementation you want. However, here are a few restrictions :

- The only allowed functions from the `libc` are the ones that wrap system calls (and don't have C++ equivalents !)
- Any solution to a problem `MUST` be object oriented.
- Any non-explicitly allowed library is implicitly forbidden.
- Any value passed by copy instead of reference or pointer `MUST` be justified, otherwise you'll lose points.
- Any non-`const` value passed as parameter `MUST` be justified, otherwise you'll lose points.
- Any use of a pointer when a reference would have been more suitable `MUST` be justified, otherwise you'll lose points.
- Any member function or method that does not modify the current instance not `const` `MUST` be justified, otherwise you'll lose points.
- Koalas don't use any `C++` norm. However, any code that we deem unreadable or ugly can be arbitrarily sanctioned. Be rigorous !
- Any conditional branching longer than `(if ... else if ... else ...)` is FORBIDDEN ! Factorize !
- Keep an eye on this subject regularly, it could be modified.
- We pay a great attention to our subjects, if you run into typos, spelling mistakes or inconsistencies, please contact us at `koala@epitech.eu` so we can correct it.
- You can contact the authors by mail, see the header.
- The intranet's forum will contain informations and answers to your questions. Be sure the answer to you question is not already there before contacting the authors.

Chapitre VIII

Turn-in instructions

You MUST deliver your project on the GIT repository made available for you by the Koalab. Your repositories will be opened at most 48h after the end of registration date, as attested by Epitech's intranet. It won't be possible to register to the project or to the defense once this date is passed.

Your repositories will be closed for writing at the exact date of the project's end, as attested by Epitech's intranet. Complaining that "it wasn't 11 :42 pm on my watch" is useless. The only time that counts for us is Epitech's time because the system is automated.

Murphy's law corollary : "If you turn in your work during the last hour, something's gonna go wrong.".

Only the code on your repository will be evaluated during the defense. The documentation relative to the repositories provided by the Koalab is provided with this subject.

Good luck !