JETPACK

FLY TO THE NETWORKING WORLD

JETPACK

Preliminaries

binary name: jetpack_client, jetpack_server
language: C or C++
compilation: via Makefile, including re, clean and fclean rules

The goal of this project is to create a game like Jetpack Joyride but with a twist, it needs to be playable with two players at the same time.
There are two parts in this project: the game server and the client program.
Like for the myFTP project, you **must** use `poll(2)` to handle your network clients.

The Makefile must have at least 3 rules (in addition to the classic clean, fclean and re rules):

- ✓ a **client** rule, to compile the client program
- ✓ a **server** rule, to compile the server program
- ✓ an **all** rule, to compile both client and server programs

You can find on the intranet an archive with references binaries and maps.
You may also find some assets to help you with the graphical part of the client.

Launch options

```
Terminal
~/B-NWP-400> ./jetpack_server -p <port> -m <map> [-d]
```

```
Terminal
~/B-NWP-400> ./jetpack_client -h <ip> -p <port> [-d]
```

You must provide a `-d` flag to enable a debug mode.
The debug mode **must** print to `stdout` information about every single packet of data you send/receive (it can be formatted however you like but it must give all the informations necessary.)
The debug mode **may** print more data if you think it is needed, but you **need** to be able to explain why it is necessary.

Protocol

The protocol is entierely up to the students to design, implement and document.

It can be text based or binary based.

It **must** be TCP based (no UDP).

Check the Documentation section of that pdf for an explanation of what is expected

You **may** implement the same protocol between 2 different groups, this will allow you to test your binaries between them and to verify that the protocol is properly documented

It is **forbidden** to share code between different groups, you can only share the documentation of your protocol!

Gameplay

The following elements should be added to your game:

- ✓ **Map**
  The server **needs** to load a map from a file.
  The client **cannot** open this file, it needs to be sent one way or another by the server.
  The map format is kept to your discretion, but it **must** be documented (see the documentation section of that pdf.)
- ✓ **Waiting for players**
  The server **must** wait for at least two players to connect before the game starts.
  The game start **can** be automatic or manual, but it **needs** to never launch without at least 2 players connected.

Your server **may** handle more than 2 players, but it is not required.

- ✓ **Map limits**
  The players **must not** leave the map, neither through the top nor the bottom.
  Horizontal movements **must** always be carried out, even if the player is against the ceiling or the ground.
- ✓ **Collisions**
  When a player collides with a coin they win one point. Every coin **can** be collected by both players.
  When a player collides with an electric square they die. Players **cannot** collide with one another.
- ✓ **End of game**
  The game is over when the players come to the end of the map or when a player dies.
  If a player dies, the other player wins the game.
  If both players reach the end of the map, the one who has the better score wins.
  In all other cases, there is no winner.
- ✓ **Reconciliation**
  A client **must** not decide if it is dead or not by itself, only the server can.
  You therefore **need** to make sure that even if clients send rubbish data that the server be working as expected (think how rampant cheating is in modern online gaming.)

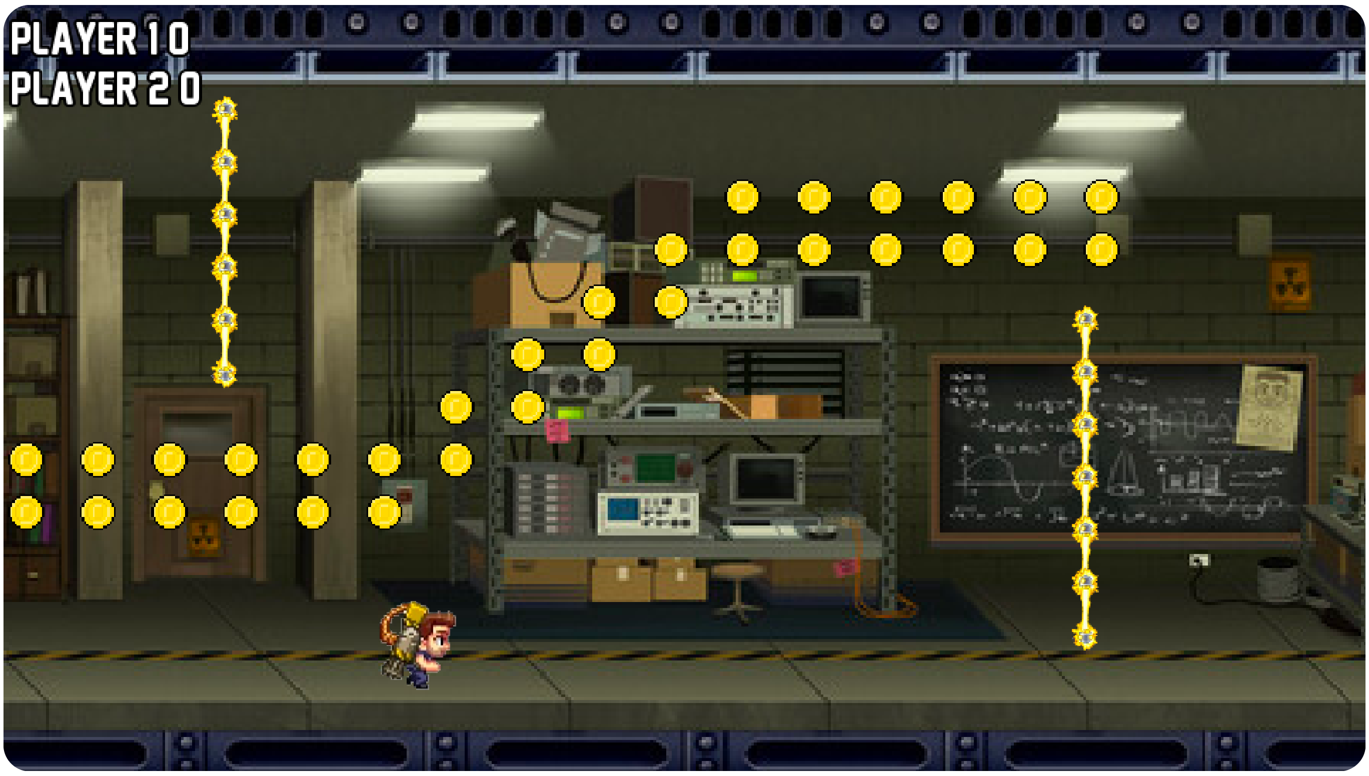
Client

Your client should handle the following:

✓ the communication with the server in a thread, which **only does communications and nothing else**
✓ the graphics interface in a second thread which draws to the screen

The network thread will only take care of updating shared game state, while the display thread will display to the screen.

Any graphical library that is available on the dump is authorized (SFML, CSFML, SDL, Irrlicht, ...)



Documentation

There is a big part of documentation on that project, please take special care of its quality

You need to document the entirety of your procotol and your map format.

Format

All the documentation must be submitted as a txt file in the format of an official RFC.
Therefore it needs to be paginated, 80 column max, and have all the required sections from an RFC (base yourself on RFC 7322 for any formatting questions.)

It **must** be at the root of your repository in a file named `doc.txt`
It is very important that this file be in your repository at the time of the delivery!

Examples

For textual protocols you can find a good documentation on RFC 595 (you normally already know about this one)

For binary protocols we suggest the Minecraft procotol community documentation (it is not an RFC format, but the way the packets are described is really nice) or RFC 6455 (especially section “5.2. Base Framing Protocol”).

v5