# B4- Synthesis Pool

B-ADM-343

# jetpack2Tek3

Server

v1.1

EPITECH.
L'ECOLE DE L'INNOVATION ET DE L'EXPERTISE INFORMATIQUE

# jetpack2Tek3

## Server

| | |
|---|---|
| **binary name**: | serverJ2T3 |
| **repository name**: | jetpack2Tek3_$YEAR |
| **repository rights**: | ramassage-tek |
| **language**: | C |
| **group size**: | 2 |
| **compilation**: | via Makefile, including re, clean and fclean rules |

- Your repository must contain the totality of your source files, but no useless files (binary, temp files, obj files,...).
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

The Makefile is shared with the client
You need a **server** rule in order to compile the server.

The goal of this project is to create a game like Jetpack Joyride.
This project deals with the server section.

## Network

Firstly, the server's network layer needs to be handled.
The server must be able to accept 2 clients and implement the following protocol:

**ID** ask for its ID (positive and unique) through the client.
*client request:* ID
*server response:* ID <value>

**MAP** ask for the map through the client.
*client request:* MAP
*server response:* MAP <width> <height> <cells>., where <cells> is a character string of length <width>*<height> that represents the map's contents ('_' for an empty space, 'c' for a coin and 'e' for an electric square)

**READY** tells the client that it has received its id and map and that it's waiting for the game to start.
*client message:* READY

**FIRE** tell the client about the change in the state of the jetpack ('0' if deactivated, '1' if activated).

*client message:* FIRE <state>

**START** tells the server that the 2 players are connected and ready and that the game is beginning.
*server message:* START

**PLAYER** sends , on the server's behalf, the state of a player.
*server message:* PLAYER <playerID> <x> <y> <score>

**COIN** tells the server that a player has found a coin.
*server message:* COIN <playerID> <x> <y>

**FINISH** tells the server that the game is over and who the winner is.
*server message:* FINISH <winnerID> or FINISH -1 if there is no winner.

Each message sent by the server or the client ends with a '\n'

All invalid commands will be ignored

## Handling the Map

Secondly, the following gameplay should be added to the server:

- **Map**
  The file representing the map (given as parameter in the program) must be loaded with memory in order to be sent to the clients and used by the server.

- **Waiting for players**
  The server must wait for the two players to connect, retrieve their respective IDs and the map and then send the message, READY. The server will then send the message, START.

- **Frames**
  The base frame is in the lower left-hand corner, the abscissa axis is horizontal and positioned toward the right. The ordinate axis is vertical and positioned toward the top.

- **Start-up**
  At the beginning of the game the players are positioned at the middle of the left-hand edge. Their coordinates are $(0, \frac{height}{2})$.

- **Magnitude**
  The players are considered to have a 1x1 size and a mass of 1 (in order to apply gravity).

- **Movements**
  The players are subject to a horizontal movement of 5 squares pr second, in the direction of increasing abscissas. The players must also be subject to gravity (passed as parameter in your binary). Activating a player's jetpack will reverse gravity for him/her.

- **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, he/she wins one point and the coin is removed. When a player collides with an electric square, he/she dies. 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.

# Prototyping and examples

```
Terminal
~/B-ADM-343> ./serverJ2T3 -p <port> -g <gravity> -m <map>
```

The server should not be blocking
Only one select (or a similar function) is authorized within your program (completely unrelated to non-blocking sockets, which are forbidden so no *fcntl(s, O_NONBLOCK)*

Here's an example of communication between the client and the server:

```
First client connection
1 <-- ID
  --> ID 3
1 <-- MAP
  --> MAP 8 4 __________e______e_______cccc__
1 <-- READY

Second client connection
2 <-- ID
  --> ID 21
2 <-- MAP
  --> MAP 8 4 __________e______e_______cccc__
2 <-- READY

Beginning of game
--> START
--> PLAYER 3 0 0 0
--> PLAYER 21 0 0 0
...
2 <-- FIRE 1
...
--> COIN 3 2 0
--> COIN 21 2 1
...
--> PLAYER 3 2.3 0 1
--> PLAYER 21 2.3 1.2 1
...
--> COIN 3 3 0
--> FINISH 3
```

Here's an example of a map file:

```
_____e_____
_____e_____cccccc_____
_____e_____ccccccc_____eeeeeeeeeee_____
_____e____cc_____
_____e___cc_____e_____cccc_____
_____cc_____e_____cccc_____
_____ccccccc_____e_____
_____cccccc_____e_____eeeeeeeeeee_____
_____e_____
_____e_____
```