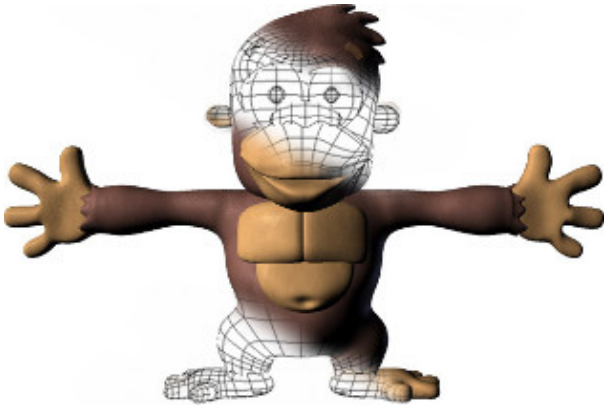
B1 - C Graphical Programming

B-MUL-100

My Runner

Who said Geometry Dash ?



1.5

My Runner

binary name:	my_runner
repository name:	MUL_my_runner_$ACADEMIC_YEAR
repository rights:	ramassage-tek
language:	C
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).

In this project, you are asked to make a small video game based on the rules of a finite Endless Running Game.

The basic rules for the my_runner are as follows:

- the player is a character who runs in a map you took as parameter.
- enemies and obstacles must appear on the opposite side to the player position.
- the player can use the space bar to jump and avoid obstacles and enemies.
- as the player runs, a score, which will be displayed, will increase.
- when the player dies or finishes the map, the score is displayed inside the window.

As you previously did, being able to manage inputs from the user and to display animated sprites are key skills when you want to develop basic games. Furthermore, you will need to manage various new game making skills such as:

- reading a file describing an environnement and use it to render a visual.
- manage basic game physics.
- manage multiple entities at the same time.

You will reuse your work for other video game projects, so think about it to make it the most scalable way.

Your project should of course fit the requirements below but we are expecting it to be more than just a technical demonstration: we want you to think about it as a real game. As such, try to have a consistency between your sprites, your sounds effects and your world.

This project will seem harder than the previous one to you.
Take your time and work on it iteratively and it can be done !

REQUIREMENTS

MUST

- The window **must** be closed using events.
- The program **must** manage the input from the keyboard.
- The program **must** contain animated sprites rendered thanks to sprite sheets.
- The program **must** contain moving (rotating, translating, or scaling) elements.
- The program background **must** include a parallax scrolling with at least 3 objects moving at different speeds.
- The program **must** take a file in argument which will contain the map of your current game.
- The program **must** display a score which is update regularly during the game.
- The program **must** have an end whether it's a victory or a defeat.
- The program **must** handle basic game physics (jumping / falling / hitting a wall).
- The program **must** have at least one music and one sound effect.

SHOULD

- The program **should** have randomly spawning enemies.
- Animations and movements in your program **should** not depend on the speed of your computer.
- Animations and movements in your program **should** be timed by *sfClock* elements.
- Your window **should** stick between 800x600 pixels and 1920x1080 pixels.
- Your window **should** have a limited frame rate such that it can be compute without lagging.
- The program **should** accept the "-h" option, then display usage of the program.
- Possible user interactions **should** be explicitly explained in a usage.

COULD

- The program **could** have several different levels.
- The program **could** have a main menu.
- The program **could** have a pause menu.
- The program **could** store the highest score made.
- The program **could** have an infinite mode where you generate randomly the map till the player dies.
- The program **could** let the user customize its character.
- The program **could** have bonus / malus which gives / removes points or advantages to the player.

The size of your repository (including the assets) must be as small as possible. Think about the format and the encoding of your resource files (sounds, musics, images, etc.). An average maximal size might be 15MB, all included.

Gameplay

While the requirements don't go deep into the gameplay, we are still asking you to make a video game.

A basic example of what we are expecting is a simple game where we could score points by avoiding obstacles (either by jumping or crouching) while our character is running towards them.
If you make an endless level the score should also based on how long the player can keep avoiding obstacles.

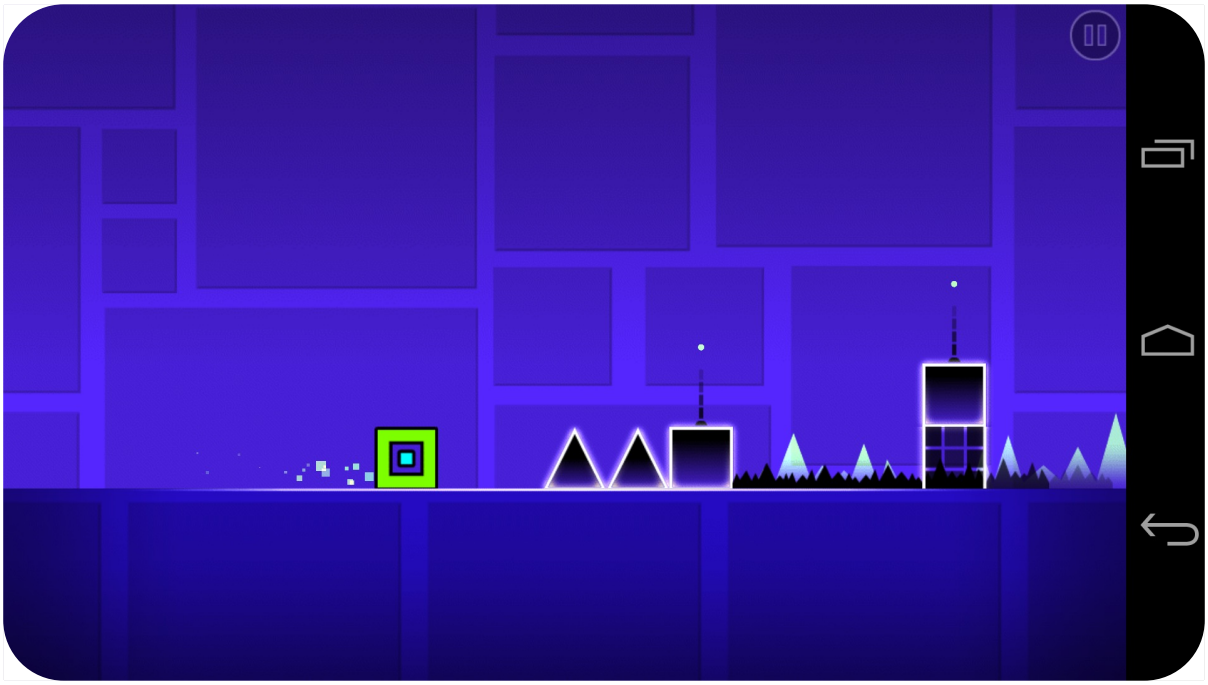
Map example

Here is a example of a map where spaces are the sky, 1 are solid ground and 2 are spikes that the player can't touch.

For the example, we use 3 to describe blocks with a different sprite. You are free to use any kind of map formating as long as it's describe in a .legend file.

```
Terminal
~/B-MUL-100> cat map/map1
                3
          223   1

11111111111111111111111111111111111111
11111111111111111111111111111111111111
11111111111111111111111111111111111111
```



USAGE

Terminal

```
~/B-MUL-100> ./my_runner && echo $?
./my_runner: bad arguments: 0 given but 1 is required
retry with -h
84
```

This is an example, you don't need to have the exact same usage. Keep in mind that a good usage covers the integrality of your project.

Terminal

```
~/B-MUL-100> ./my_runner -h
Finite runner created with CSFML.

USAGE
  ./my_runner map.txt


OPTIONS
  -i                launch the game in infinity mode.
  -h                print the usage and quit.

USER INTERACTIONS
  SPACE_KEY         jump.
```

AUTHORIZED FUNCTIONS

Here is the full list of authorized functions.

from the C library
- write
- malloc
- free
- memset
- rand
- srand
- (f)open
- (f)read
- (f)close
- getline

from the CSFML library
All functions

from the math library
All functions

Any unspecified functions are de facto banned, except for bonus features.