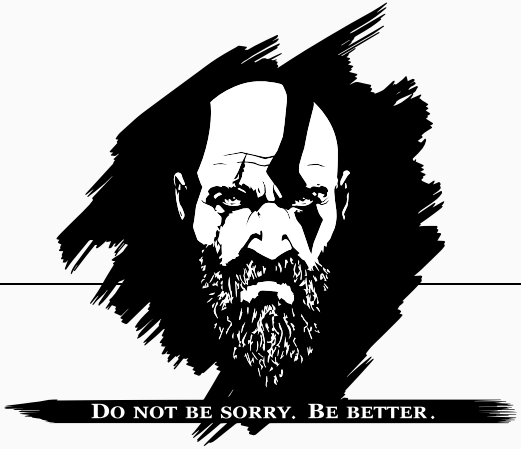
Rush C - Presentation

ACU 2019 Team



Copyright

Rules

- You must have downloaded your copy from the Assistants' Intranet <https://intra.assistants.epita.fr>.
- This document is strictly personal and must **not** be passed on to someone else.
- Non-compliance with these rules can lead to severe sanctions.

The Rush

What is it?

The free implementation of a video game.

What you know

After a few months, you saw:

- How you can choose different implementations, with *Malloc*.
- How to work in teams on a complex project, with *Formula One* and *Raytracer*.

What to do now?

It is now time to put it all together.

A video game is a good way of expressing your creativity *and* showing off your newly acquired skills.

Subject & Expectations

- Purpose & Possibilities
- Core/Additionnal features
- What we expect

Design & Tools

- Code design
- SDL2
- Game design

Subject & Expectations

A platformer

- Gameplay mechanics where the player interacts with encountered obstacles in uneven terrain.
- Action-based game, more details in the subject.
- A lot of features in the subject to match your game ideal.

Figure 1: Famous platform games

Figure 2: Famous platform games

CF & *AF*: Definition

You will be marked on the features that you will implement.

Two types of features:

- *Core features* (CF):

  Expected, give a minimum mark

- *Additional features* (AF):

  Not a bonus, needed for a good mark.

Only the features listed in your README will be evaluated.

Simple, to fit your game idea.

- A main character.
- Two maps.
- One ennemy and one other way to harm the player.
- Game over.

- Implementing only the core features will not give you a perfect grade
- They are **NOT** bonuses
- Additional features have a level ranging from *Trivial* to *Hard*
- Do not trick us with only *Trivial* or *Easy* features

This is the moment to let your imagination run wild! **Have fun**, implement as much as you can and impress us!

You have a good idea? Add it as an additional feature!

- If the feature requires arrangement to be made to test it during your defense, post a news.

  ex: Testing multiplayer with several computers.

- Otherwise, just add the feature to your feature list in your README

Be aware that it is a rush, so you must focus on what you can do in this short time.

[VGR][NAF] Multiplayer (2 coop)

Greetings,

We would like to implement a 2 player coop mode, playable from different computers.

With this, the players would have fun on the same map with greater boss and more complex mechanics.

This would add some challenge to the game!

Regards,

–
Lambda Ing1

- Evaluated solely during a defense.
- The mandatory part isn't enough to get a good grade!

Very important for this project.

We expect it to explain:

- How to build your game.
- How to play your game:
  - Controls
  - Little introduction of the story, the environment, obstacles, boss, etc.
- The different sources used (for the sprites, sounds and other resources).
- The exhaustive list of the features (CF & AF) to evaluate.

Every game is going to be a unique one.

Thus, explain well how to play yours!

- 38 hours to create a video game.
- Hard work expected: not much time to sleep.
- Group of 3 students.

Subject & Expectations

- Purpose & Possibilities
- Core/Additionnal features
- What we expect

Design & Tools

- Code design
- SDL2
- Game design

Design & Tools

One of the main challenges of your rush.

You must organize yourself properly. Do not hesitate to dedicate 2 hours at the beginning of the rush to decide on:

- What game you want to make.
- How you want to structure it.
- How you will split the tasks.

Code design: A challenge

- Think about which features you wish to implement beforehand.
- Plan for the future.

Do not hesitate to ask for advices from the assistants for your code architecture.

- Refactor the fields of your structures
- For example, your character and the enemies will most likely need a position, speed, life...
- These fields don't have to be in both structures, create a structure containing those fields in common instead

- Restrict the initialisation of a variable to one instance.
- Useful for Main Window and other objects where you need global access.

```
static SDL_Window *screen = NULL;

SDL_Window *get_screen(void)
{
    if (!screen)
        screen = SDL_CreateWindow(...);
    return screen;
}
```

Warning

- In most case it is not OK to use them.
- Not thread safe.

Game State

- A cleaner way to put all the context you need into a single structure initialized once.
- You then just have to pass this structure as a parameter to most of your functions.
- This structure could look like that (think before copying this):

```
struct GameState
{
    SDL_Renderer *renderer;
    struct Map *map;
    struct Player *player;
    ...
};
```

SDL2: Simple DirectMedia Layer

- Hardware abstraction library.
- Cross-platform API (Windows, Linux, Mac, Android, IOS).
- Written in C.
- Multiple language bindings (C, C++, Go, OCaml, Python, Rust...).
- Free for personal and commercial use.
- https://wiki.libsdl.org/
- link with `-lSDL2`.

- Image
- Mixer
- net
- ttf

To use SDL2:

- You must include and link it.
- You must pay attention to dependencies!

Example: `LDLIBS=-lSDL2 -lSDL2_image`

- You should use SDL2.
- A tutorial will be given on the intranet.
- A good documentation: https://wiki.libsdl.org/

Warning

Many online tutorials are outdated.

```c
struct GameContext game;

// Simple loop that runs as long as the game is on
while(game.is_playing)
{
    // get user input (key pressed, windows closed, etc)
    struct Input in = get_current_input();

    // compute new positions, sates, etc
    update(&game, event);

    // draw the changes on the screen
    render_frame(game);
}
```

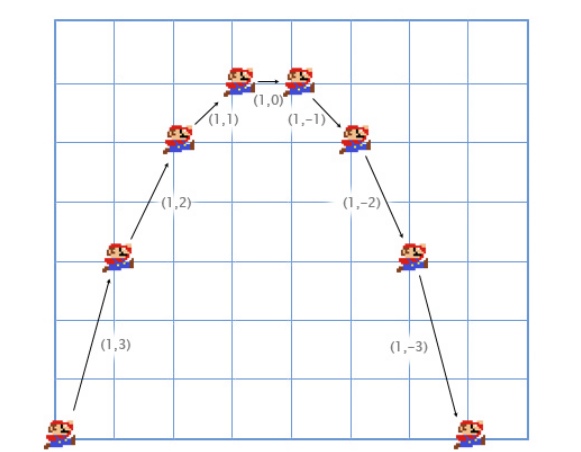


Figure 3: Jump Parabola

Formula:

- F = ma
- a = F/m (Sum of all forces)
- dv/dt = a
- dx/dt = v
- **v = a * dt**
- **x = v * dt**

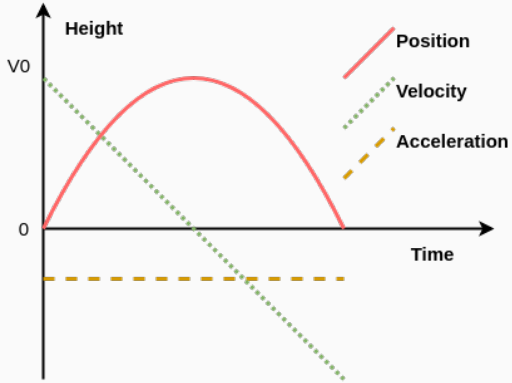


Figure 4: Position, Velocity and Acceleration

In this case, we only consider the gravity. We willingly omit friction, etc.

FPS (Frame Per Second)

Correctly compute your frame (characters movements, etc) according to your frame rate / fps.

- Basic mechanism of a game.
- Without delta timing the more fps you can compute on your computer, the faster your game runs.
- Synchronize the speed at which the elements of the game are moving with their theoric speed.

```c
double delta_time(uint64_t *last_update_time)
{
    // Get sdl update frequency
    double frequency = SDL_GetPerformanceFrequency();

    uint64_t last    = *last_update_time;
    uint64_t now     = SDL_GetPerformanceCounter();

    // Updates the last time the frame has been computed
    *(last_update_time) = now;

    // The delta time is in seconds
    return (now - last) * 1000 / frequency;
}
```

Using Delta time to compute new position

```
struct Object
{
    vect2_t acc;
    vect2_t vel;
    vect2_t pos;
};

void update(struct Object *o, double delta_time)
{
    // Modify forces if needed (change gravity, etc)
    // Compute new acceleration if forces have changed

    o->vel += o->acc * delta;
    o->pos += o->vel * delta;
}
```

- There are other ways to take the delta time into account (for example inserting an artificial sleep between frames to match a given frame rate, or using *SDL_RENDERER_PRESENTSVSYNC*)
- Important to have knowledge of the concept.
- Not mandatory but will be taken into account for the grade.

Conclusion

Take a few hours to plan your project.

Start simple, then add features.

Split the work well.

Project

- You **should** use SDL2 for this project.
- It is well-documented.
- A lot of things are already done.
- You also can use OpenGL for your rendering or shaders.
- Do not forget to have fun.

Newsgroup : `assistants.projets`, `[VGR]` tag.

Deadline : November 11th, 11:42 a.m.

As usual:

- Your project **must** comply with the coding-style, especially during this project since your code will be reviewed.
- Cheating will be **strongly** penalized.
- You will not be helped if you don't have a `Makefile` or if you didn't attempt to debug.
- **Your README is important.**

When calling an ACU, you must explain:

- Your debugging steps.
- What you expected to happen.
- What actually did happen.

Any questions?