LY Benjamin

21 ans

Ynov Bordeaux Campus

B3 – Game Programming

benjamin.ly97@ynov.com

RAPPORT- MARIO LIKE C++ SFML

2021 - 2022

Proget de jeux vidéo en C++ dans le cadre du programme de la B3 Game Programming à Ynov Bordeaux Campus

TABLE DES MATIERES

Table des matières

RAPPORT - MARIO LIKE C++ SFML

Présentation du jeux

Il s'agit d'un jeux mario en 2D développé en C++ avec la SFML (une bibliothèque graphique).
La version du jeux mario que je me suis inpsiré est celui du Super Mario world sur SNES sorti en 1990.

Image tiré du jeux original :

Image tiré de mon mario like :

RAPPORT - MARIO LIKE C++ SFML

Mécanique du jeux

Principal concept :

- Le joueur incarne Mario et doit aller jusqu'à la fin du niveau pour finir le jeux
- Il se déplacer vers la gauche et la droite et sauter
- Rebondi sur la tête des ennemies et les tues
- Si le joueur se fait touché par un ennemie, alors il devient petit, mais c'est game over s'il est déjà petit
- Peut manger un champignon (au collision) pour devenir grand

Les inputs :

- « Q » et « D » pour se déplacer
- « S » pôur s'accroupir
- « Espace » pour sauter

Architecture du projet

Le projet se divise en 2 partie distinctes :

- 1 dossier contenant les fichiers codes du jeux
- 1 dossier contenant l'éxécutable du jeux, les DLL ainsi que les ressources (images, musique, …Etc)

Un script python et un setup du compilateur g++ y sontt également dans le dossier principal pour pouvoir compiler le code.

Architecture du code

Dans le dossier principal du dossier contenant les fichiers code, il y a 2 dossiers particuliers :

- DataGame
- Game

Contenu du fichier main.cpp :

- Dossier Game

2 classes essentiels à la construction du jeux :

- Game, boucle de jeu se divisant en 3 partie successive : Update -> Perform move sprite -> Collision
- GameEngine -> permettant de faire tourner le jeux (mise en place du niveau, déplacement des personnages, les update, les collisions, ..etc)

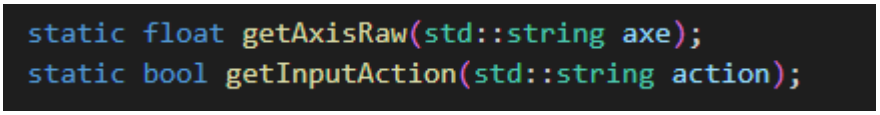
Il y aussi une classe GameOverEngine permettant la mise en place du Game Over.

La classe GameEngine (et GameOverEngine aussi) est utilisé dans la classe Game qui est utilisé dans le fichier main.cpp.

- Dossier system

Le dossier System contient l'ensemble du code correspondant au « Project Setting » comme on peut le retrouver dans Unity ou Unreal Engine dans lequel on peut configurer les inputs du jeux.

Il y a une classe Input contenant des fonctions static permettant de définir les inputs qui pourront être utiliser dans le jeux. (Toujours inspiré de Unity et Unreal Engine)



Exemple :

```
bool Input::getInputAction(std::string action)
{
    if (action == "Jump")
    {
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space))
            return true;
    }

    return false;
}
```

```
if (Input::getInputAction("Jump")
{
```

Ici, la touche « Espace » a été mappé à l'action « Jump ».

Les touches directionnels « Z Q S D » ont été mappé pour correspondre aux axes horizontal et vertical.

Ci-dessous, la fonction getAxisRaw retourne un float lorsque la touche correspondant à l'axe donnée en argument est pressé.

```
xMov = Input::getAxisRaw("Horizontal") * speedMove * deltaTime;
```

« Q » retourne -1 et « D » retourne 1.

Si aucune de ces touches ne sont préssées ou que les 2 touches sont préssés alors 0 sera retourné.

- Dossier DataGame

Corresppondant à l'ensemble des éléments du jeux, que ça soit graphique ou gameplay.

Une classe contenant tous ces éléments est utilisé dans le GameEngine qui met en intérraction ces éléments dans l'univers du jeux.

Les éléments principals sont :

- Mario
- Les ennemies : Goombas
- Item : Champignons, Pièce
- Les éléments UHD (ces éléments là sont fixés à l'écran), ex : compteur de pièces
- La Map, avec un système de parallax (relié à l'emplacement de la caméra)
- Les collisions
- Les boîtes de collisions (callés sur toute l'infrastructure de la map)

Les éléments comme mario et les ennemies découle d'une class mère appélé Actor.
Héritant tous de variable commun tel que :

- Sprite
- Spee move
- Hp
- Etc…

Ainsi que de fonctions communs.

Gestions des collisions

Dans le dossier DataGame, il y a un dossie Collision contenant la classe Collision.

Celle-ci possède des fonctions statics qui pourront être appelé pour vérifier s'il y a collision entre 2 sprites.

Exemple :

```
//////
bool PixelPerfectTest(const sf::Sprite &Object1, const sf::Sprite &Object2, sf::Uint8 AlphaLimit = 0);
```

```
if (Collision::PixelPerfectTest(mario->footHitbox->getSprite(), actors[i].sprite->getSprite()))
{
```

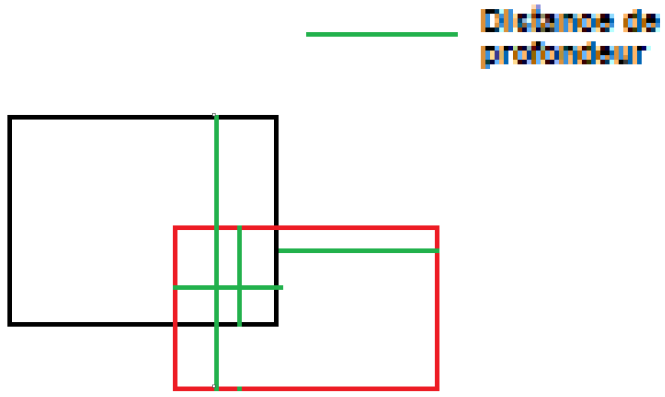
Il y a une fonction particulière permettant d'annuler la collision entre 2 sprites (square) :

```
// Cancel collision by all side
void NotTrigger(sf::Sprite &sprite1, sf::Sprite &sprite2);
```

En effet, un sprite (donc un square, ou encore un quadrilatère) possède 4 côtés.

Mais par quel côté, vecteur normal, faut-il utiliser pour repousser la sprite à annulé la collision.

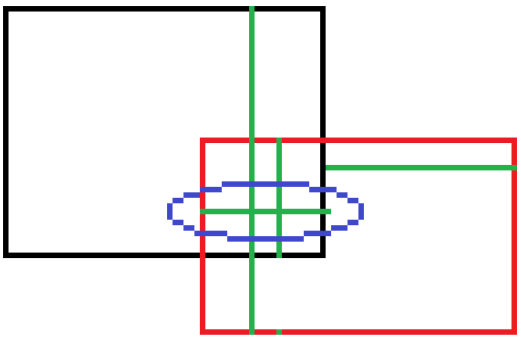
Explication mathématique ci-dessous :



Lorsqu'il y a collision, on calcule les 4 distances de profondeurs.

La plus petite déterminera le vecteur normal à utiliser pour annuler la collision.

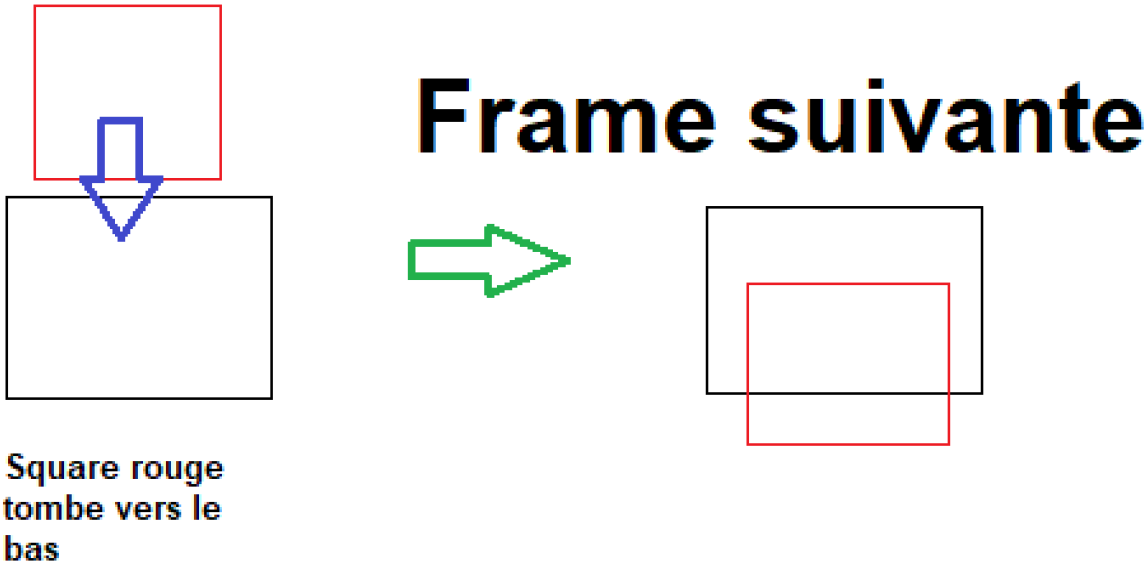
Exemple :



Imaginons qu'ici la plus petite distance de profondeur est celle vers la droite (entouré en bleu), donc le vecteur normal utilisé sera (1, 0);

Cependant il y a des soucies de gestions de collision lorsque les sprites se déplacent très vite.

Exemple de cas de problème de collision :



L'annulation de la collision devrait se faire par le haut, mais ici elle se fera par le bas, ce qui pose problème.

Le square rouge traversera alors le square noir.

Ce problème survint en particulier sur Mario, sur son saut. En effet plus il est haut dans le vide et plus sa vitesse de chute sera élevé.

Pour tenter de résoudre ce probème :

- J'ai placé 4 boîtes de collision sur Mario

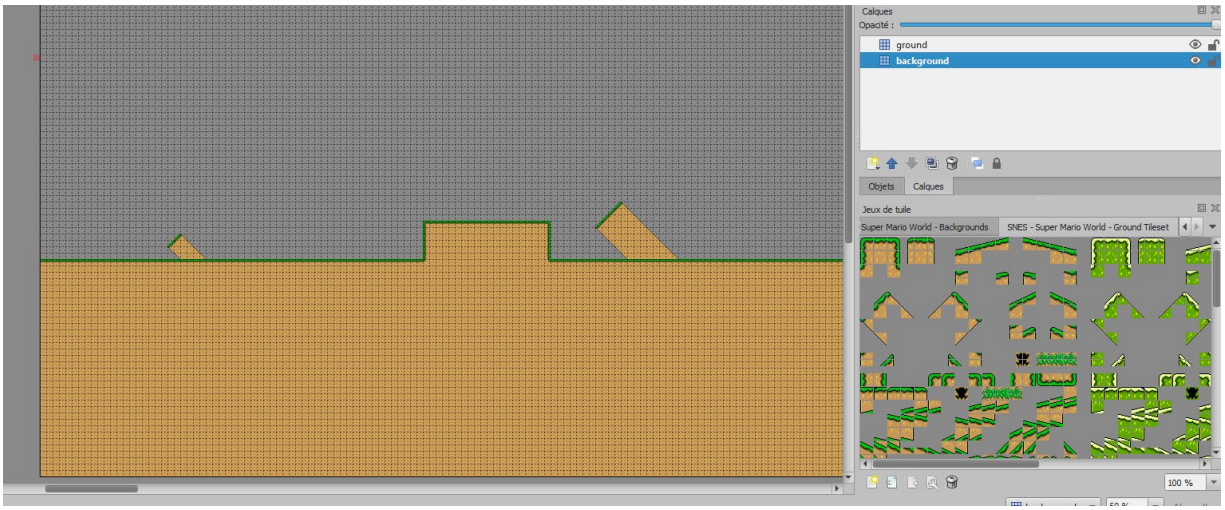
- Chaque boîte de collision annulera les collisions par le côté correspondant
- Celui du bas détecte qu'il est au sol et permettra si Mario peut sauter ou non.
- Celui au dessus de sa tête permettra de détecter les collisions au dessus, en particulier sur les blocks (celle contenant un champignon)

Certe il y a toujours quelques problèmes de collisions.

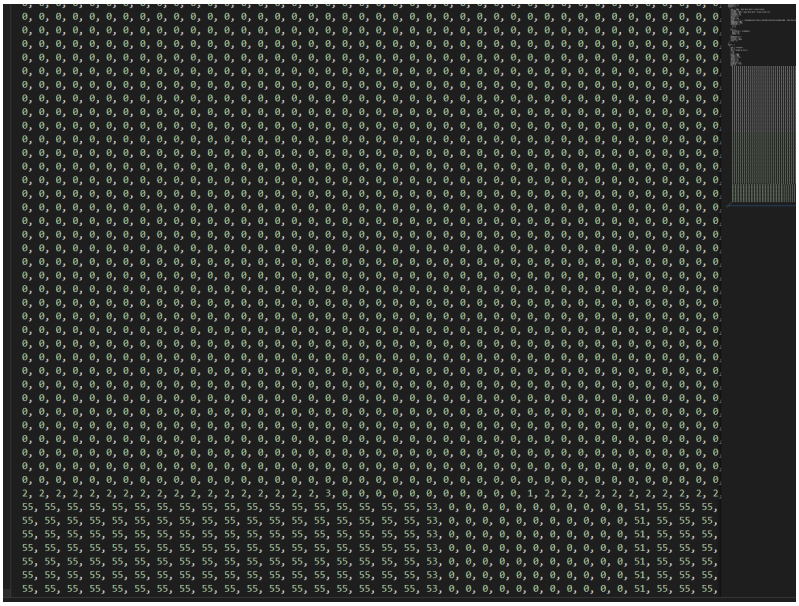
La Map

La Map a été réalisé avec un system de Tile Map avec l'utilisation du logiciel Tiled :

Possibilité de peindre le level. J'exporte ensuite en un fichier lua pour l'exploité par la suite.



Dans le fichier lua précèdement exporter, je récupère les coordonnées des Tile set pour les implémenter et former ma map que j'ai édité sur le logiciel.



Sur le site officiel de la SFML, il y a un tuto pour implémenter les tile Map :

https://www.sfml-dev.org/tutorials/2.5/graphics-vertex-array.php

Un code permettant cette implémentation y est également proposé, que j'en ai repris.

Conclusion

Ce jeux a été réalisé par moi-même tout seul. C'était un choix de le faire tout seul.

Le plus gros challenge de ce projet était la gestion des collisions faisant appel à des notions mathématiques.

La gestion de la camera avec la parallax a été aussi un challenge.

J'ai bien aimé faire ce projet de Mario Like.

Mais j'aimerais ne plus jamais faire de la SFML, c'est-à-dire sans moteur graphique.

Le placement des ennemies ainsi que des autres éléments ont été placé plus ou moins à l'aveugle.

Sur Unity, j'aurais fait ce projet en 1 jour … ou 2.