Jeu de VS Fighting

2019-2020

Ziehl Benjamin, Simon Arthur, Barbillon Pierre

Table des matières

UNIVERSITÉ DE LORRAINE | IUT Saint-Dié-des-Vosges

1) Introduction

Le but de ce projet est de créer un jeu type « Versus Fighting », c'est à dire un jeu où deux personnages s'affrontent l'un contre l'autre. Le jeu sera créé sans l'aide de logiciel tel que Unity et sera codé en C++ en utilisant la librairie SFML, utilisant OpenGL.

Les contraintes qui nous sont imposées sont de travailler en mode 'Agile', notre projet sera décomposé en plusieurs sprint. Un sprint est un objectif à réaliser en peu de temps et dont on attend une application 'fonctionnelle' dès la fin du premier, ensuite chaque sprint suivant visent à améliorer cette application déjà fonctionnelle jusqu'à ce qu'on obtienne le produit attendu.

Les contraintes que nous nous sommes imposées sont de créer le jeu de A à Z en utilisant un langage avec lequel nous sommes familier mais que nous n'avons pas étudier à l'IUT . Cela implique aussi de réaliser nous même les graphismes et les sons pour la création du jeu.

Ce projet sera d'une grande aide afin d'en apprendre plus sur la conception d'un jeu vidéo, il sera aussi le témoin de nos capacités à mener un projet en groupe et en autodidacte.

2) Démarche du travail effectué

A) présentation

Dans un jeu vidéo, la partie la plus importante est le gameplay, c'est la principale raison pour laquelle un jeu est bon et reçoit l'attention des joueurs. Le gameplay concerne l'intrigue du jeu et comment il est joué, s'il est bon, un joueur se sentira bien en jouant, c'est aussi simple que cela, et pour qu'il soit bon, vous devez comprendre ce que le joueur veut. C'est pourquoi nous avons mené une interview avec un joueur de jeu de combat pour comprendre ce qu'il attendait d'un jeu de combat.

La deuxième chose dont nous avons besoin d'un jeu vidéo et de bons graphismes, des effets sonores et pas de bugs, si vous perdez dans un jeu à cause de bugs, vous vous sentirez très frustré et cela ruinera votre expérience du jeu, il en va de même pour les graphismes et les effets sonores, si ces derniers ne conviennent pas ou / et sont de mauvaise qualité, il peut devenir désagréable à jouer au jeu.
Attention ! Une mauvaise qualité ne signifie pas «non réaliste», vous pouvez avoir un jeu de combat avec des bonhommes en bâtons que le joueur considérera de bonne qualité parce que le jeu est fluide, agréable à jouer et à regarder.

Exemple d'un jeu 'simple' mais qui a eu du succès grâce à un gameplay innovant

Mes camarades et moi avons une passion pour les jeux, nous avons donc décidé d'en créer un pour notre PT3, le projet concerne un jeu de combat où les professeurs de notre IUT se battraient.
Nous voulions en faire un jeu <versus Fighting>, où seuls deux adversaires se battent entre eux avec une barre de vie:

Street Fighter : figure iconique des jeux type VS Fighting

Nous voulions également incorporer des coups spéciaux qui refléteraient le professeur, comme un lancé de M&M's pour Boris Davin, un professeur qui les échange avec ses étudiants.

Bien sûr, c'étaient des idées que nous avons eu au départ mais lorsque nous avons commencé à coder le jeu, nous avons compris à quel point c'était vraiment difficile et nous avons réfléchi à d'autres solutions et sommes arrivés à la conclusion qu'il fallait simplifier le jeu au maximum, puis étoffer le contenu. Tout d'abord, un personnage avec des mouvements de base et un arrière-plan, puis, lorsque cela fonctionnera, des mises à jour pourraient être effectuées.

B) aspects technique

Nous avons utilisé SFML pour créer notre jeu qui utilise le langage C ++, ainsi que Box2D, un moteur physique qui nous a aidé à créer la gravité et les collisions dans notre jeu.

C++

SFML

Box2D

C ++ est le langage qui est presque toujours utilisé dans la conception de jeux car il permet au développeur de contrôler les ressources utilisées par le matériel, ce qui signifie qu'avec C ++, vous pouvez utiliser efficacement la puissance de l'ordinateur du joueur.

Code

```
// definition de la barre de vie
sf::RectangleShape hp(int x,int y) {

    sf::RectangleShape hp;
    hp.setSize(sf::Vector2f(pv, 40)); // Taille
    hp.setPosition(x, y); //Position sur l'écran
    hp.setFillColor(sf::Color(100, 250, 50));

    hp.setOutlineColor(sf::Color(0, 0, 0)); //Couleur du Contour
    hp.setOutlineThickness(4); //Taille du contour

    return hp;
}
```

Méthode hp : barre de vie des personnages

La particularité de cette méthode ‘hp’ est qu’elle prend en paramètres deux entiers qui définissent la position de la barre de vie (cette dernière étant déclaré dans le main) et qu’on définit sa longueur grâce

à l'attribut 'pv' , de la classe personnage. Ce dernier est initialisé à 300 donc la barre de vie fait 300 pixels de long et lorsqu'un personnage prend des dégâts, une méthode 'setHp' va réduire cette attribut, ce qui va aussi réduire la taille de la barre de vie.

```
if (p1.getHP() == 0 or p2.getHP() == 0) {
    sf::Sprite spriteKo;
    sf::Texture texture;
    texture.loadFromFile("C:/Users/pc 1/source/repos/PT3/Ko.png");
    spriteKo.setTexture(texture);
    spriteKo.setPosition(sf::Vector2f(370, 10));
    window.draw(spriteKo);
    window.display();
    std::this_thread::sleep_for(std::chrono::milliseconds(10000));

}
```

Condition d'arrêt d'une partie

Ici, nous avons une condition d'arrêt qui s'exécute dans une boucle 'while' qui est actif tant que le jeu est ouvert, elle détecte si les points de vie de l'un des personnages est à 0 et si c'est le cas, affiche une image 'Ko' et met fin a la partie. (Sur la capture elle met simplement le programme en 'sleep')

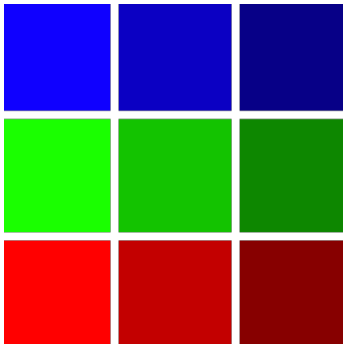
```
if(p1.getX() < p2.getX() && p1.getX() + 80 > p2.getX() or p1.getX() < p2.getX()+80 && p1.getX() + 80 > p2.getX()+80) {
    p1.setHP(10);
    sound.play();

}
```

Exemple de gestion des collisions

Enfin, une condition elle aussi exécuté dans la boucle du main qui vérifie à chaque fois si les dimensions d'un personnage sur l'axe des abscisses (p1.getX() et p2.getX()) se trouve dans celle d'un autre personnage, si c'est le cas alors c'est qu'il y a collision.

Cette fonction est générale et ne prend pas en compte s'il y a un mouvement d'attaque ou de défense, elle est surtout utile pour faire des tests, ici dans la condition, on réduit les pv du personnage 1 'p1.setHP(10)' et on joue un son qui est un bruitage de cri de douleur.

Pour les graphismes, nous avons utilisé Piskel. C'est un logiciel d'animation 2D pixélisé en ligne très simple et agréable à utiliser, comme nous le verrons plus tard, j'ai dû m'entraîner un peu afin de me familiariser et travailler plus efficacement avec ce dernier…



logo de Piskel

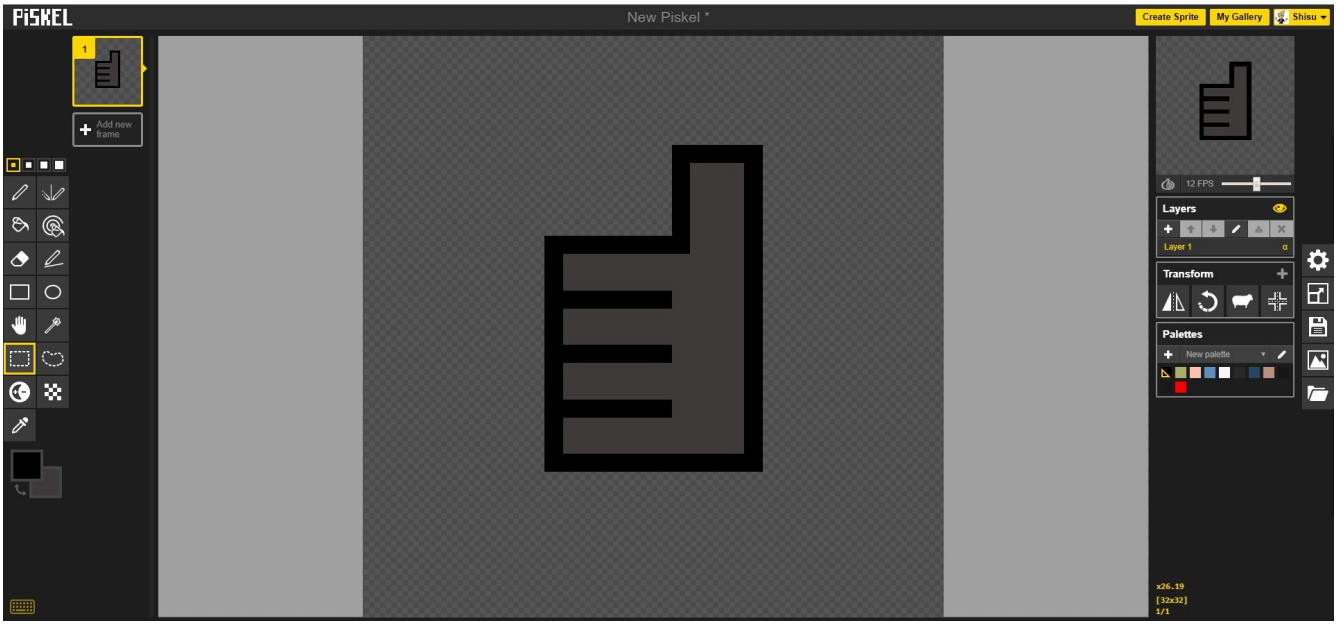
Piskel offre tous les outils nécessaires pour créer une animation 2D fluide dans les arts du pixel, contrairement aux autres logiciels de dessin, Piskel remplace la quantité par l'efficacité, ce que je veux dire, c'est qu'au lieu de donner au dessinateur un grand nombre d'outils et de fonctions qui peuvent être vraiment pratiques mais difficiles et longues à apprendre, il propose juste ceux que vous utiliserez le plus souvent, comme un crayon cubique, une gomme, un modèle carré…

Pour l'implémentation des animations, l'utilisation de la bibliothèque Thor est primordiale. Cette bibliothèque utilise SFML, et est très utilisée pour simplifier le processus d'animation. Pour créer une animation complète, prenons l'exemple d'une mouvement de garde (lorsque le personnage ne bouge pas). On déclare d'abord une animation. On a ensuite besoin d'ajouter une par une toute les Frames de l'animation, avec un temps relatif. Ensuite, on la construit avec un temps absolu. On peut ensuite la jouer par exemple avec la condition qu'aucune touche de clavier ne soit appuyée.

```
thor::FrameAnimation walk;
thor::FrameAnimation stance;
thor::FrameAnimation back;
thor::FrameAnimation jump;
thor::FrameAnimation down;
thor::FrameAnimation sPunch;
thor::FrameAnimation downK;
thor::FrameAnimation wPunch;
thor::FrameAnimation kick;
```

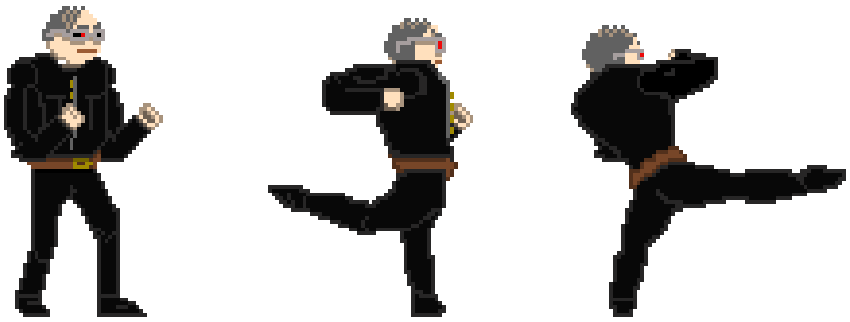
```
walk.addFrame(1.0f, walk1);
walk.addFrame(1.0f, walk2);
walk.addFrame(2.0f, walk3);
walk.addFrame(1.0f, walk4);
```

```
if(sf::Keyboard::isKeyPressed(sf::Keyboard::Right)){
    sprite.move(2.0f, 0.0f);
    if(animator.getPlayingAnimation()!="walk"){
        animator.playAnimation("walk", true);
    }
}
```



Capture d'écran de l'interface du logiciel Piskel

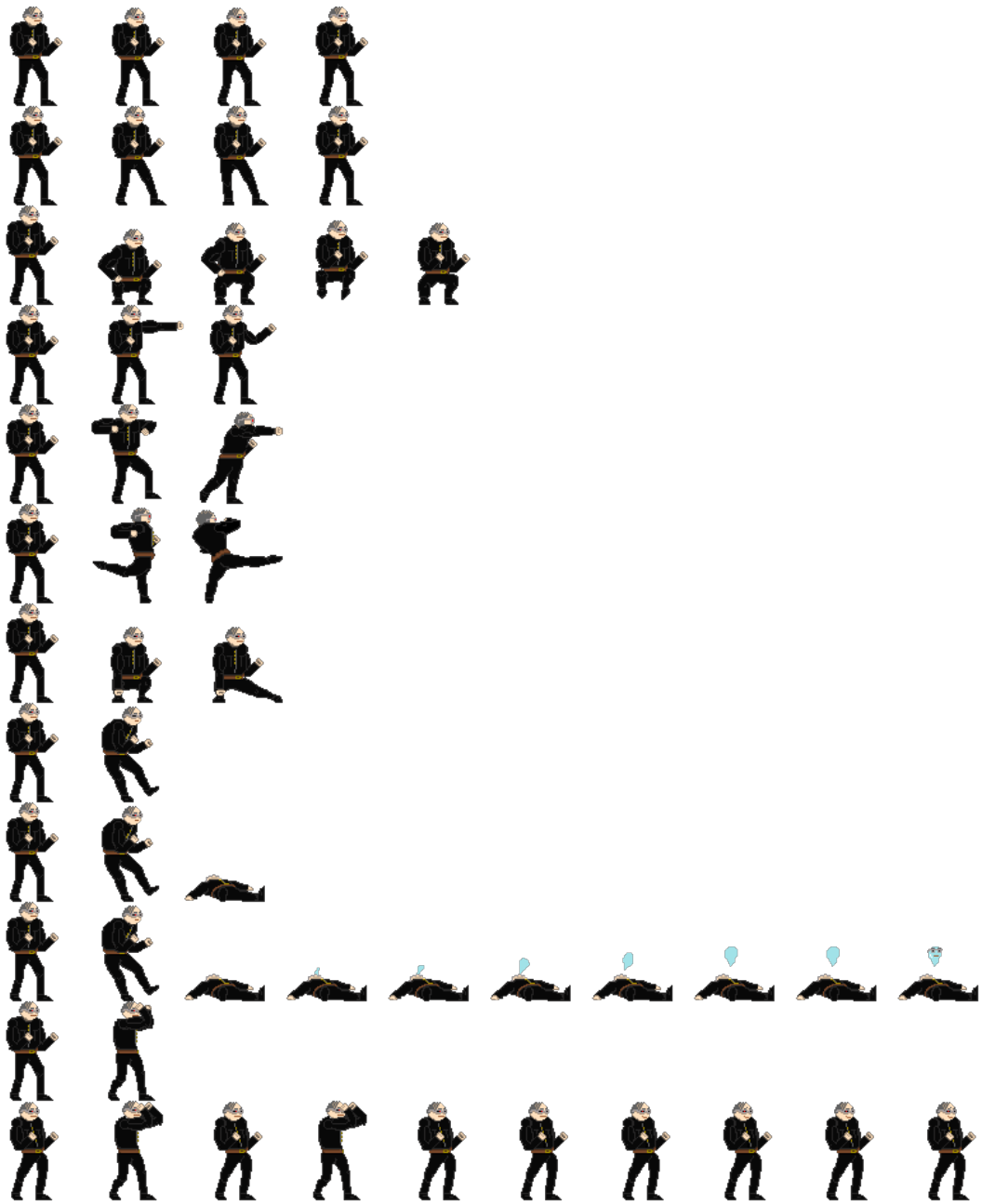
En haut à droite, vous pouvez voir à quoi ressemble votre dessin et si vous dessinez plusieurs images, la fenêtre jouera l'animation au rythme que vous avez sélectionné (0 à 24 ips), une fois que vous avez terminé avec votre animation, vous pouvez enregistrer votre projet ou l'exporter sous forme de fichier png, vous pouvez alors choisir comment vous voulez que vos multiples frames (images) soient collés ensemble.



Exemple d'exportation au format png : animation d'un coup de pied

Ici, on voit un personnage effectuer un coup de pied, les images on étés collées en largeur permettant de représenté un des mouvements de la SpriteSheet.

Si l'on prend plusieurs mouvements et qu'on en fait une Spritesheet (Une images regroupant tout les mouvements d'un personnage) alors on obtient ceci



SpriteSheet de notre combattant

C) Organisation

Il a fallu beaucoup de temps pour vraiment démarrer la programmation de notre jeu car c'était un gros projet, et nous ne savions pas vraiment par où commencer.

Nous avons d'abord choisi le langage informatique que nous voulions utiliser (C ++), puis la bibliothèque (SFML) et le moteur physique qui nous feraient gagner BEAUCOUP de temps et d'ennuis car il serait trop difficile d'en créer un nous-mêmes.

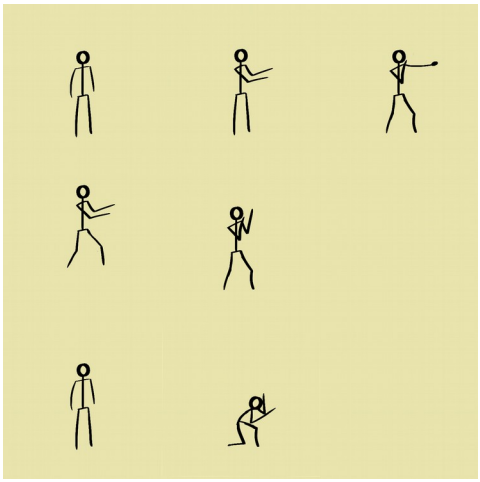
Ensuite, nous avons séparé le travail de manière à ce que l'un fasse les graphismes et la documentation tandis que les deux autres travaillent sur les parties techniques.

L'élève chargé des graphismes et de la documentation avait tout de même sa part de code à réaliser.

Pour la partie documentation, le travail s'est fait en fonction des documents a rendre, les plus urgents en premier.

La programmation était un peu plus délicate car il fallait avoir une partie des graphismes (SpriteSheet & background) afin de commencer à les implémenter.

Afin de commencer à coder le plus tôt possible, nous avons utilisé stick man pour les tests.



3) Résultats

Une introduction se lance avec le programme, présentant le titre du jeu mis en scène dans une animation, puis,une seconde fenêtre s'ouvre et une musique se lance en même temps que le jeu, on voit alors deux personnages face à face à l'IUT de Saint-Dié, deux barres de vie, le combat commence et chaque coup reçu est suivi d'un cri de douleur jusqu'au Ko final qui indique la fin du combat, la victoire d'un joueur, la défaite de l'autre.

Les mouvements ont tous été programmés sur THOR mais nécessitent encore d'être implémentés dans le code principale.

4) Bilan

Un projet de création de jeu vidéo requiert normalement un coût élevé et beaucoup de temps, n'ayant eu ni le temps ni l'argent ce fut un projet rude à mener et bien évidemment loin d'être complété (le contenu proposé étant faible pour être un jeu complet) mais l'expérience était certainement enrichissante du fait que nous ayons dû nous occuper de tout, y compris les sons et graphismes qui étaient hors du domaine informatique.

Nous avons aussi pu tester nos compétences d'apprentissage en autonomie et découvrir les avantages et inconvénients de travailler à plusieurs sur un même projet informatique, notamment les difficultés que pose l'exécution d'un code sur des systèmes d'exploitation différentes (entre Windows et Linux) .

Somme toute nous sommes satisfaits de notre travail, voir les personnages que l'on a créé, s'animer en fonction des touches sur lesquelles on appuie donne un sentiment d'accomplissement et bien qu'il manquait encore un peu de travail pour réussir a tout faire fonctionner, nous avons réussi a créer un jeu de combat crédible .

Si l'expérience était a refaire, nous serions plus a même de cerner le projet et l'ampleur qu'il représente. Nous pourrions alors définir et répartir les tâches plus efficacement, de plus, nous avons été plutôt flexibles sur le travail en mode Agile donc ce serait un point intéressant à revoir car nous n'avons pas vraiment fonctionner par 'sprint' bien qu'il y aie effectivement eu plusieurs étapes dans notre projet, donc revoir notre approche de la méthode Agile pourrait améliorer nos résultats.

5) Perspective

On peut tout à fait imaginer pour la suite de ce projet une application des solutions vu en étude de l'existant avec un décor interactif (Stick Fight) ou un gameplay innovant (Lethal League).

Un enrichissement du Gameplay en ajoutant de nouveaux mouvements, personnages, décors qui donnerait au jeu plus de légitimité et une meilleur longévité (c'est à dire qu'avec plus de contenu, le jeu sera jouable plus longtemps).

On pourrait aussi revoir le code et les solutions informatique utilisé afin d'en trouver de meilleurs qui pourraient apporter plus de fluidité et/ou dynamisme au jeu, en gérant les données efficacement pour éviter par exemple l'utilisation d'espace inutilement grâce a un programme reproduisant les fonctions d'un Garbage Collector.

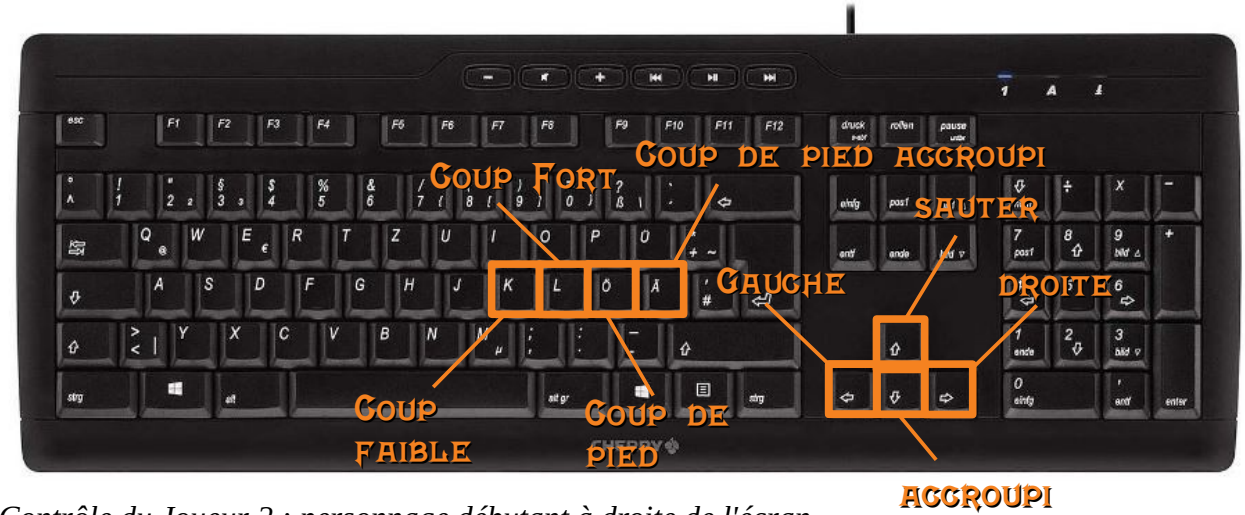
En partant du principes que le jeu est passé par les étapes vues précédemment, on pourrait considérer sa commercialisation sur le marché (en reversant des royalties à l'IUT bien sûr) sur des plateformes tel que Steam.

Après sa mise en vente et suivant son succès, on peut aussi imaginer proposer des mises à jour ajoutant plus de contenu, il est assez courant d'en voir sur les jeux vidéos mais ce qui est rare, surtout dans un jeu de combat, c'est une diversification du Gameplay via des mises à jours, on pourrait donc instaurer un concept nouveaux dans les jeux de VS fighting qui est une 'Meta' c'est à dire une version du jeu ou certains personnages ou combos sont plus efficaces que d'autres et qui peut changer d'une Mise à jour à l'autre, affectant la manière de jouer des joueurs .

6) Annexes

Mode d'emploi

Il faut cliquer sur l'exécutable du jeu identifiable par le nom '*Mortal Te4ch-R.exe*' , une fois le jeu lancé on utilise les touches comme indiqué ci-dessous :



Contrôle du Joueur 2 : personnage débutant à droite de l'écran

Coût et budget

Il n'y a pas de budget pour ce projet, la seul source de coûts est l'utilisation d'appareils électrique entraînant une usure de ces derniers ainsi qu'une utilisation d'électricité mais il nous est impossible d'en identifier l'impact monétaire.

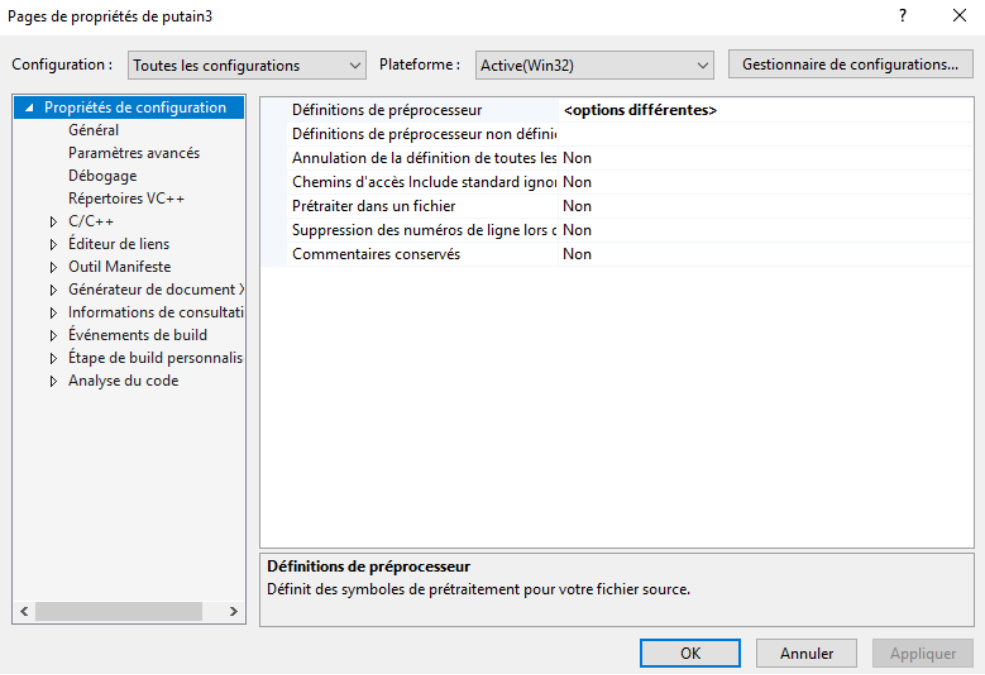
Documentation technique

Pour installer SFML sur l'IDE et le système d'exploitation souhaité, veuillez vous referez à la documentation que nous avons produit :

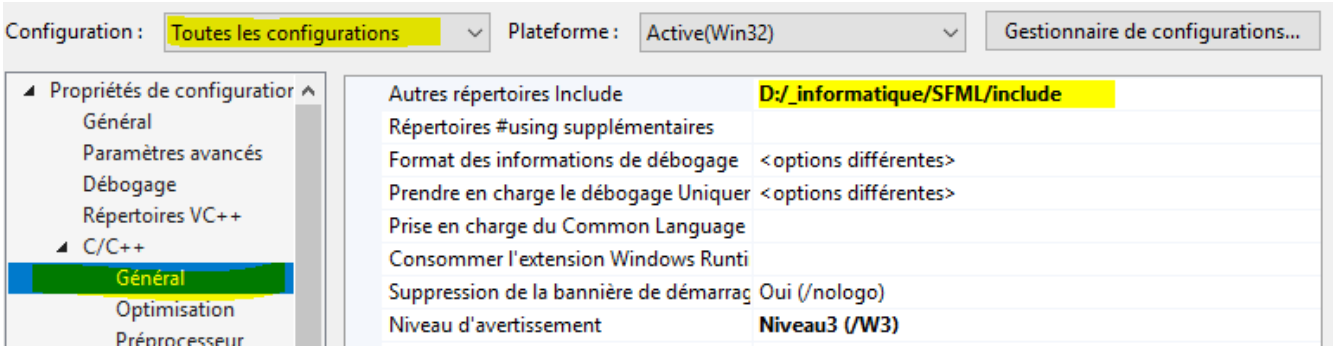
https://www.sfml-dev.org/download-fr.php - télécharger SFML

Installation SFML avec Visual Studio

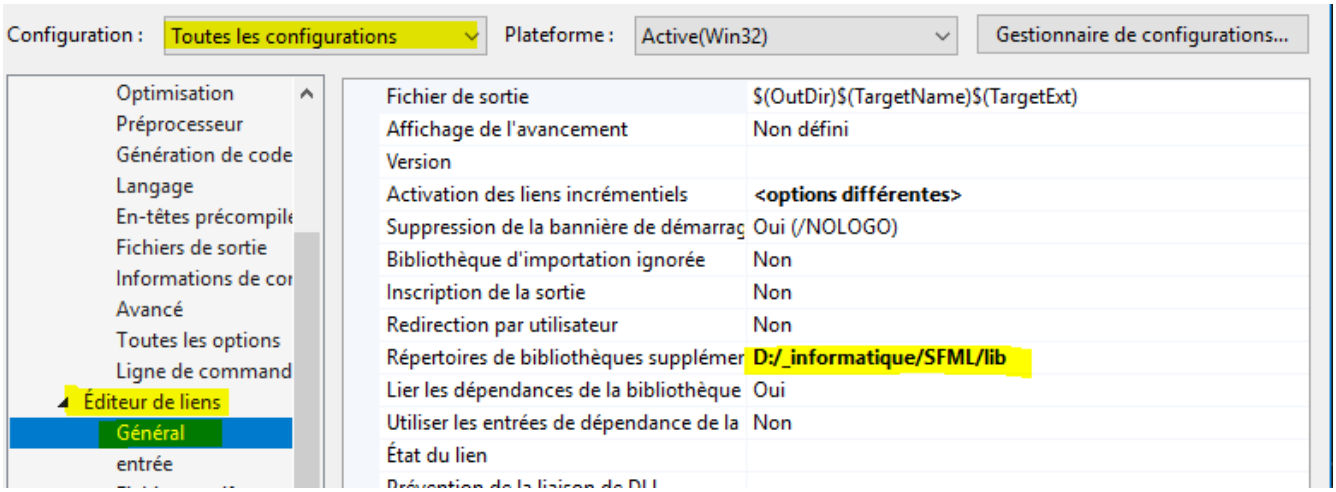
Pour Visual Studio, après avoir créer un projet on se rend dans : Projet > propriétés de 'nom du projet'



Il faut ensuite se rendre dans C/C++ > Général et donner le chemin vers le fichier SFML que vous avez télécharger plus tôt ainsi que son include.

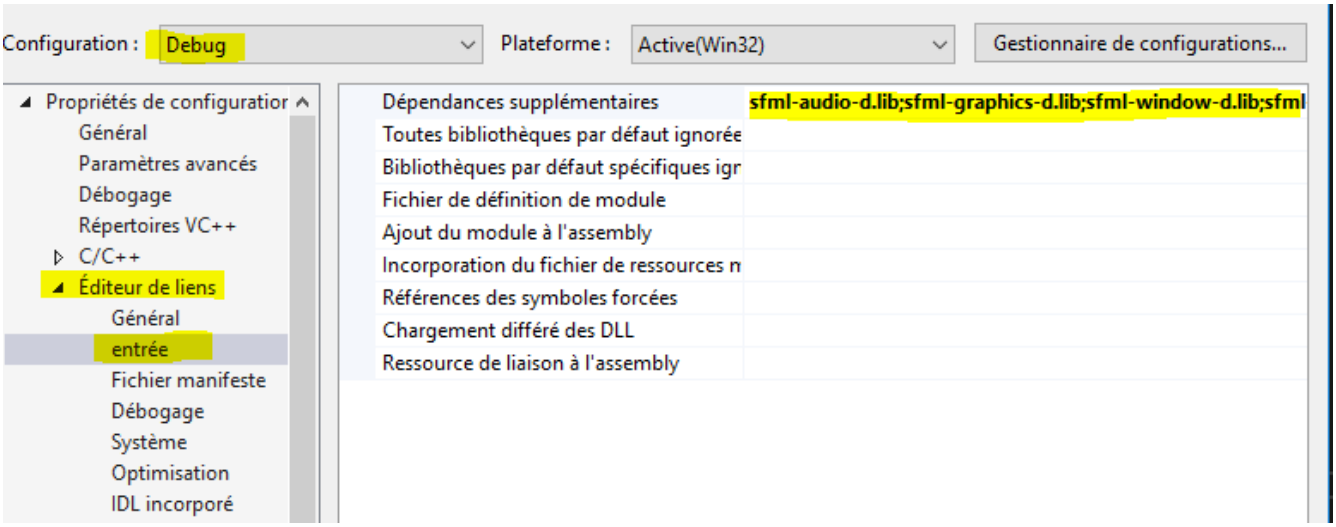


faite attention à la configuration, nous verrons qu'elle changera plus tard
Il faut aussi importer le dossier lib d'SFML, pour cela



On se rend dans Editeur de liens > Général et on donne le chemin jusqu'au dossier 'lib' d'SFML dans la partie 'Répertoires de bibliothèques supplémentaire ' .

On se rend ensuite dans les entrée

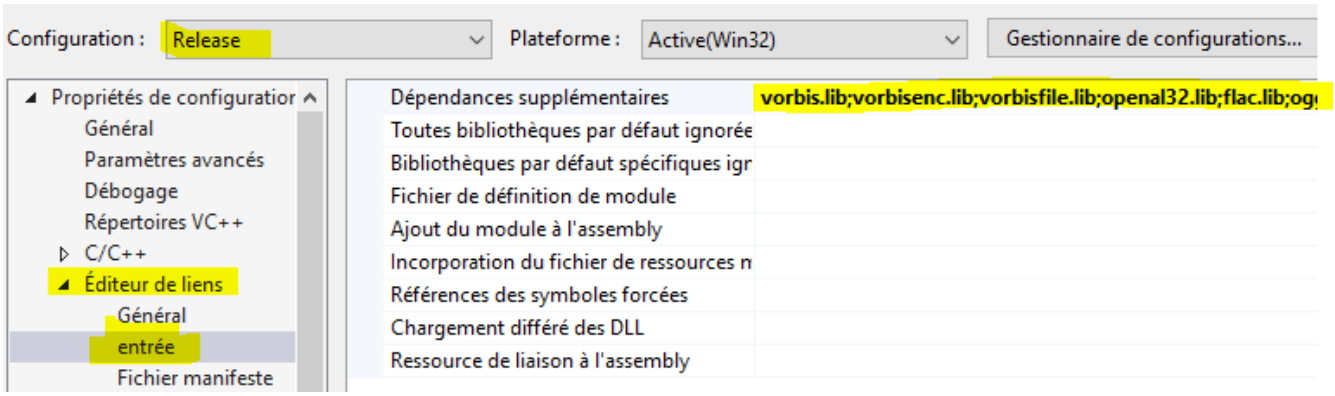


Et on y ajoute les dépendances avec la librairie SFML, pour cela , ajouter avec un copier/coller :

sfml-audio-d.lib;sfml-graphics-d.lib;sfml-window-d.lib;sfml-system-d.lib;

On sépare chaque éléments par un point-virgule, a noter que la configuration est ‘debug’ d’où le -d

On fait pareil pour la configuration release à quelques exceptions prêt, le -d est remplacer par -s et il faut ajouter d’autres dépendances, notamment pour pouvoir utiliser la librairie audio :

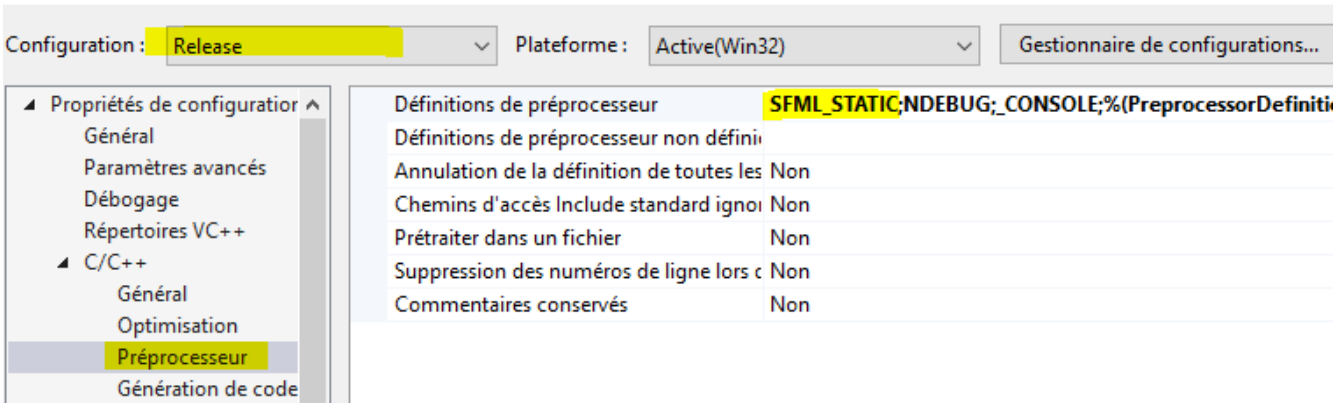


Voici les dépendances a copier :

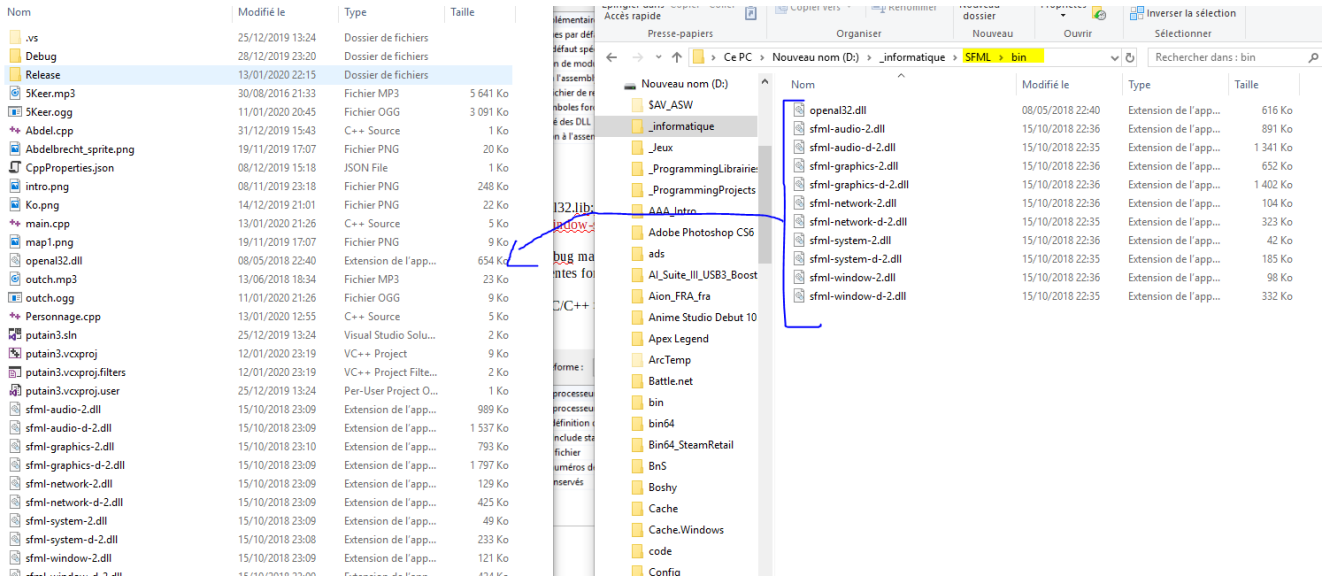
vorbis.lib;vorbisenc.lib;vorbisfile.lib;openal32.lib;flac.lib;ogg.lib;winmm.lib;opengl32.lib;freetype.lib;sfml-audio-s.lib;sfml-graphics-s.lib;sfml-window-s.lib;sfml-system-s.lib;

La partie en rouge est la même que pour debug mais avec le -s a la place du -d et le reste sont les fichier a importer pour faire fonctionner les différentes fonctions de SFML.

Enfin, il faut ajouter SFML_STATIC dans C/C++ > préprocesseur et cela uniquement dans la configuration ‘Release’ .

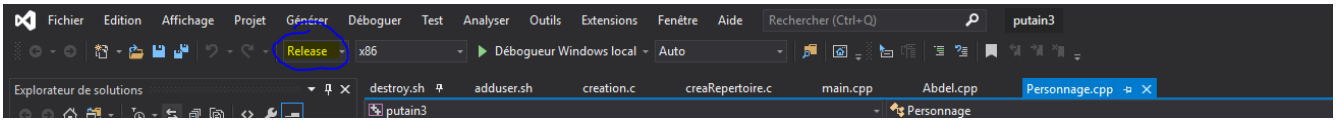


Au cas où il y aurait toujours des erreurs lors de la compilation, effectuer l’étape suivante :



Copier tous les fichiers du dossier bin de SFML dans le répertoire de votre projet.

Lorsque vous lancé le code, pensez a bien mettre Visual Studio en ‘Release’

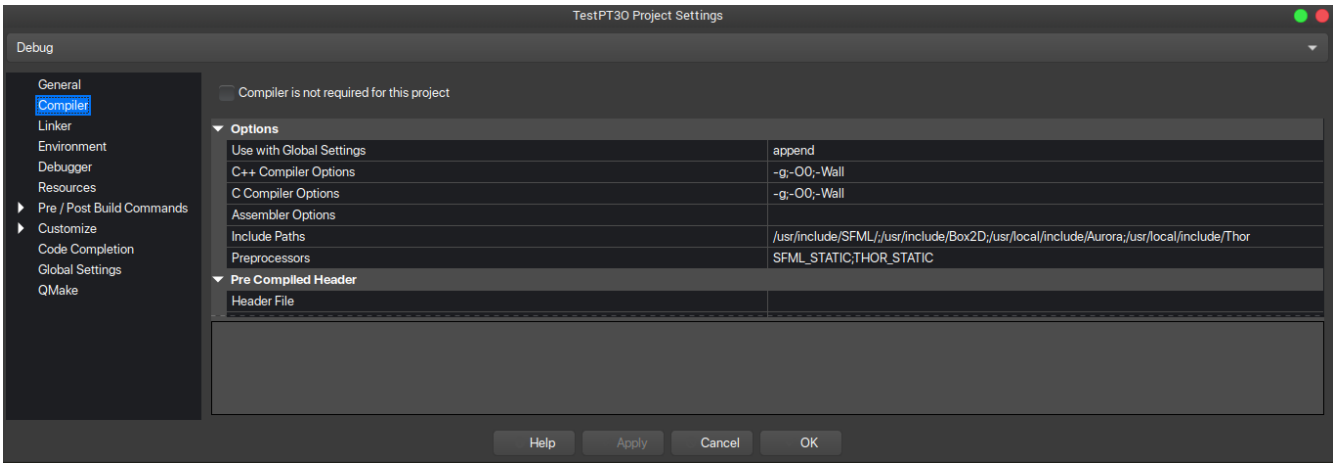


Installation de SFML sur Linux

Si la version de SFML que vous souhaitez installer est disponible dans les dépôts officiels, alors installez-la simplement avec votre gestionnaire de paquets. Par exemple, sous Debian vous feriez :

sudo apt-get install libsfml-dev

Ensuite il faut suivre des étapes similaires à celle pour l'installation de SFML sur Visual Studio, en ajoutant les chemins des lib, include , sfml-graphics, ect …, sans oublier SFML_STATIC et THOR_STATIC .



Installation de THOR sur Linux

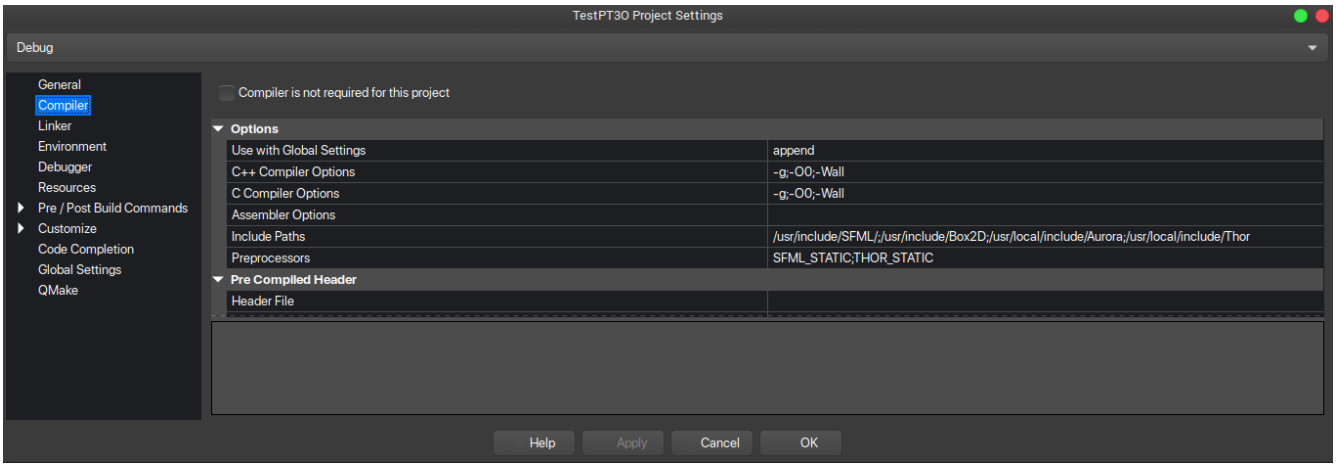
https://bromeon.ch/libraries/thor/download/index.html - télécharger THOR

Pour installer Thor sous linux, il faut prendre la version 2.0 Linux 64bits make, installer le contenu du dossier include dans /usr/local/include/

Il faut copier le contenu de lib dans /usr/local/lib/

Une fois ces 2 étapes faites, Thor est correctement installé

Pour setup l'IDE (ici codeLite) Il faut rajouter les dossier où se situent SFML et Thor pour les includes, et rajouter SFML_STATIC et THOR_STATIC dans les paramètres de préprocesseur



Pour la partie Linker (édition des liens en fr je crois), il faut rajouter le chemin des lib de Thor et SFML, ainsi que les librairies (sfml-graphics etc, sans oublier thor).

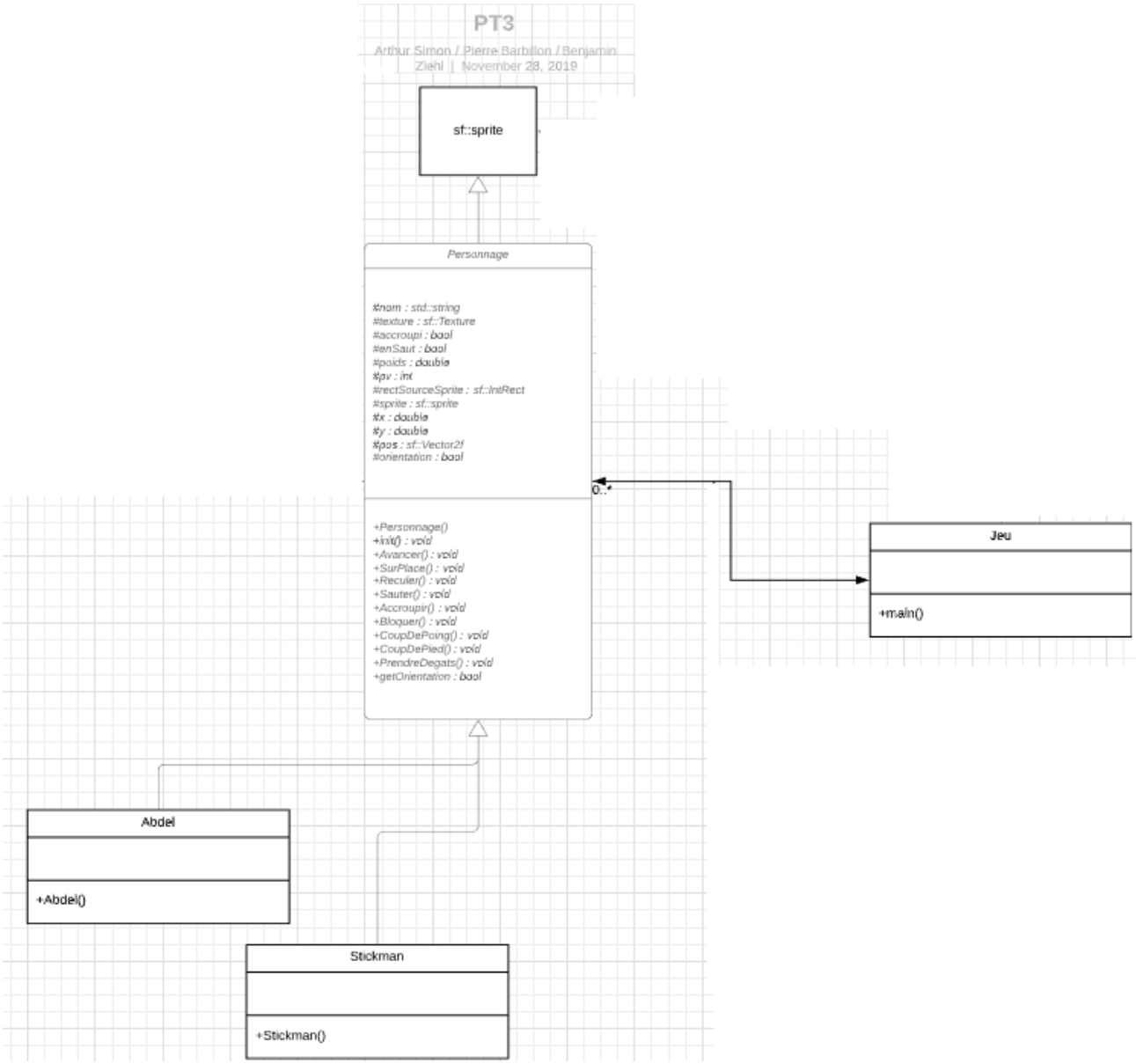
Une fois ces deux parties de l'IDE paramétrées, la compilation pourra être effectuée correctement sans soucis.

Dans le cas où votre programme n'a pas fonctionné après avoir suivi notre documentation, veuillez vous référez aux tutoriels sur le site de SFML et Bromeon(pour THOR) :

https://www.sfml-dev.org/tutorials/2.5/index-fr.php - voir rubrique 'Démarrer' pour l'installation.

https://bromeon.ch/libraries/thor/tutorials/index.html - tutoriels d'installation et d'utilisation de THOR

Diagramme de classe



Code Principale

Classe Personnage

```
#include <SFML/Graphics.hpp>
//#include "Moteur.cpp"
#include <string>
#include <thread>
#include <cstdlib>
#include <iostream>


#pragma once

class Personnage : public sf::Sprite {
public:
	Personnage() : pv(300), enSaut(false), accroupi(false), poids(0), rectSourceSprite(0,
80, 80, 80), sprite(texture, rectSourceSprite), x(0), y(0), pos(x, y) // Constructeur
	{

	}

	void init() {

		sprite.setOrigin(40, 0);

		if (orientation == false) {
			x = 200;
			y = 370;
			sprite.setScale(2, 2);

		}
		else if (orientation == true) {
			x = 770;
			y = 370;
			sprite.setScale(-2, 2);
		}
		pos.x = x;
		pos.y = y;
	}

	// definition de la barre de vie
	sf::RectangleShape hp(int x,int y) {

		sf::RectangleShape hp;
		hp.setSize(sf::Vector2f(pv, 40)); // Taille
		hp.setPosition(x, y); //Position sur l'écran
		hp.setFillColor(sf::Color(100, 250, 50));
```

```
	hp.setOutlineColor(sf::Color(0, 0, 0)); //Couleur du Contour
	hp.setOutlineThickness(4); //Taille du contour

	return hp;
}

// utiliser pour modifier les pv lorsqu'un personnage prend des dégâts
void setHP(int degat) {
	pv = pv-degat;
}

int getHP() {
	return pv;
}

void Avancer() { //Avance toujours vers l'adversaire
	if (orientation == false)
		pos.x += 10;
	else
		pos.x -= 10;

	if (rectSourceSprite.left == 160)
		rectSourceSprite.left = 0;
	else {
		rectSourceSprite.left += 80;
	}
	sprite.setTextureRect(rectSourceSprite);
	sprite.setPosition(pos.x, pos.y);
}

void SurPlace(Personnage p2) {

	if (orientation == false) {
		if ((this->pos.x) - 75 > p2.pos.x) {
			this->orientation = true;
			sprite.setScale(-2, 2);
		}
	}
	else {
		if ((this->pos.x) + 75 < p2.pos.x) {
			this->orientation = false;
			sprite.setScale(2, 2);
		}
	}
	sprite.setPosition(pos.x, pos.y);
}

void Reculer() {
	if (orientation == false)
		pos.x -= 5;
	else
		pos.x += 5;
	if (rectSourceSprite.left == 0)
```

```
                rectSourceSprite.left = 160;
        else {
                rectSourceSprite.left -= 80;
        }
        sprite.setTextureRect(rectSourceSprite);
        sprite.setPosition(pos.x, pos.y);
}

void Sauter();
void Accroupir();
void Bloquer();

sf::Sprite CoupDePoing() {


        if (rectSourceSprite.left == 1000)
                rectSourceSprite.left = 0;
        else {
                rectSourceSprite.left += 500;
        }
        sprite.setTextureRect(rectSourceSprite);
        return sprite;
}

sf::Sprite CoupDePied();


bool getOrientation() {
        return this->orientation;
}

//Permet de récupérer la position du personnage sur l'axe des abscisses
double getX() {
        return pos.x;
}

sf::Sprite getSprite() {
        sprite.setTextureRect(rectSourceSprite);
        sprite.setPosition(pos.x, pos.y);
        return sprite;
}

void TestWait() {
        sf::sleep(sf::milliseconds(10000));
}

void TestThread() {
        printf("yo");
}

protected:
        std::string nom; //Nom du personnage
```

```
	sf::Texture texture; //Toute la table de texture, qui sera ensuite transformée en
différents sprites
	bool accroupi; //false = Debout / true = Accroupi
	bool enSaut; //false = au sol / true = en l'air
	double poids; //poids du perso, afin de gérer sa gravité, la puissance de ses coups, etc
	int pv; //point de vie, démarre tjr à 100
	sf::IntRect rectSourceSprite; //Pour pouvoir le modifier sans le réintialiser à chaque
fois
	sf::Sprite sprite;
	double x;
	double y;
	sf::Vector2f pos;
	bool orientation; //false = vers la droite / true = vers la gauche

};
```

160 lignes

Classe Abdel (hérite de Personnage)

```
#include <SFML/Graphics.hpp>
#include "Personnage.cpp"
#include <string>

class Abdel : public Personnage {
public:
	Abdel(bool orient) {
		orientation = orient;
		nom = "Abdelbrecht";
		sf::Texture texture;
		texture.loadFromFile("C:/Users/pc 1/source/repos/PT3/Abdelbrecht_sprite.png");
		this->texture = texture;
		this->poids = 50;


	}

};
```

13 lignes

Main

```
#include <SFML/Graphics.hpp>
#include <iostream>
#include <ctime>
#include <string>
#include <cstdlib>
#include <sstream>
#include <iomanip>
#include <locale>
#include "Personnage.cpp"
#include <SFML/Audio.hpp>
#include <thread>
#include <chrono>
#include "Abdel.cpp"

#include <fstream>
#include <vector>

#include <functional>


using namespace std;

void TestWait() {
	printf("sleep lancé");
	sf::sleep(sf::milliseconds(10000));
	printf("sleep stopé");
}

void action(Personnage* player, char move) {
	int i = 0;
	switch (move)
	{
	case 'r':
		if (player->getOrientation())
			player->Reculer();
		else
			player->Avancer();
		break;
	case'q':
		if (player->getOrientation())
			player->Avancer();
		else
			player->Reculer();
		break;
	}

}
```

```
int main()
{

	Abdel p1(false);
	Abdel p2(true);

	p1.init();
	p2.init();

	sf::Clock clock;
	int countdown = 5;

	sf::RenderWindow windowI(sf::VideoMode(560, 438), "M0RT4L T34CH-R");
	//définition de l'intro du jeu vidéo
	sf::Texture introTexture;
	sf::Sprite intro;
	introTexture.loadFromFile("C:/Users/pc 1/source/repos/PT3/intro.png");
	intro.setTexture(introTexture);
	//560x438 pour chaque frame de l'intro.

	intro.setTextureRect(sf::IntRect(0, 0, 560, 438));
	int introTimer = 1;
	int introMax = 23;// il y a 23 images sur l'animation de l'intro

	windowI.setActive(false);
	//lancement de l'introduction
	while (windowI.isOpen())
	{
		if (introTimer <= introMax) {
			windowI.clear();
			windowI.draw(intro);// a chaque fois, on nettoie la fenêtre et on
	redessine l'image suivante
			windowI.display();

			intro.setTextureRect(sf::IntRect(560 * (introTimer - 1), 0, 560 *
	introTimer, 438));/** cette fonction permet de définir la frame que l'on veut afficher,
	a chaque fois on va a la frame suivante :
			560 * (introTimer - 1) et 560 * introTimer définissent l'image en largeur
	tandis que 0 et 438 définissent l'image en hauteur, seul la largeur change car on
	affiche l'image suivante*/
			introTimer++;
			std::this_thread::sleep_for(std::chrono::milliseconds(250));
		}
		else {
			windowI.close();
		}
	}

	sf::RenderWindow window(sf::VideoMode(945, 647), "M0RT4L T34CH-R");
	sf::Sprite map;
```

```
sf::Event event;

sf::Texture texture;
texture.loadFromFile("C:/Users/pc 1/source/repos/putain3/map1.png");
map.setTexture(texture);

window.setActive(false);

sf::Thread aaa(&TestWait);
aaa.launch();

//définition et lancement de la musique dès l'exécution du jeu
sf::Music music;
music.openFromFile("5Keer.ogg");
music.setVolume(10);
music.play();

// définition d'un bruitage 'outch'
sf::SoundBuffer buffer;
buffer.loadFromFile("outch.ogg");
sf::Sound sound;
sound.setBuffer(buffer);

//window.setVerticalSyncEnabled(true);

while (window.isOpen())
{

	while (window.pollEvent(event))
	{


		if (sf::Event::Closed) {
			window.close();
		}

		if ((sf::Keyboard::isKeyPressed(sf::Keyboard::Right))) {
			sf::Thread thread(std::bind(&action, &p1, 'r'));
			thread.launch();
		}

		if ((sf::Keyboard::isKeyPressed(sf::Keyboard::Q))) {
			sf::Thread thread(std::bind(&action, &p2, 'q'));
			thread.launch();
		}

		if ((sf::Keyboard::isKeyPressed(sf::Keyboard::A))) {
			sf::Thread thread(&TestWait);
			thread.launch();
		}
```

```
                    if ((sf::Keyboard::isKeyPressed(sf::Keyboard::Left))) {
                        if (p1.getOrientation())
                            p1.Avancer();
                        else
                            p1.Reculer();
                    }

                    if ((sf::Keyboard::isKeyPressed(sf::Keyboard::D))) {
                        if (p2.getOrientation())
                            p2.Reculer();
                        else
                            p2.Avancer();
                    }

                    if ((sf::Keyboard::isKeyPressed(sf::Keyboard::Up))) {
                    }

                    if ((sf::Keyboard::isKeyPressed(sf::Keyboard::Down)))
                    {}
                    // Condition qui simule une collision entre les sprites des deux
personnages
                    if(p1.getX() < p2.getX() && p1.getX() + 80 > p2.getX() or p1.getX() <
p2.getX()+80 && p1.getX() + 80 > p2.getX()+80) {
                        p1.setHP(10);
                        sound.play();

                    }
                    // Condition d'arrêt du jeu, un des personnages est Ko
                    if (p1.getHP() == 0 or p2.getHP() == 0) {
                        sf::Sprite spriteKo;// def du sprite et de la texture
                        sf::Texture texture;
                        // On charge la texture avec l'image de Ko ainsi que le Sprite
                        texture.loadFromFile("C:/Users/pc 1/source/repos/PT3/Ko.png");
                        spriteKo.setTexture(texture);
                        spriteKo.setPosition(sf::Vector2f(370, 10));// positionnement de
l'image de Ko au milieu des deux barres de vie
                        window.draw(spriteKo);
                        window.display();
                        std::this_thread::sleep_for(std::chrono::milliseconds(6000));
                        window.close(); // Après 6 secondes d'arrêt, la fenêtre se ferme
                    }


                    p1.SurPlace(p2);
                    p2.SurPlace(p1);

                    window.clear();
                    window.draw(map);
                    window.draw(p1.getSprite());
                    window.draw(p2.getSprite());
                    window.draw(p1.hp(50,50)); // on dessine la barre de vie et on donne sa
position
                    window.draw(p2.hp(590, 50));
```

```
                        window.display();
                        std::this_thread::sleep_for(std::chrono::milliseconds(50));
                }
        }


        return 0;
}
```

~200 lignes

Code des animations avec THOR

```
#include <iostream>
#include <string>
#include <Thor/Thor.h>
#include <SFML/System.hpp>
#include <SFML/Graphics.hpp>
int main()
{
        int x(0);
        int y(0);
        sf::Vector2f position(x, y);
        sf::IntRect stance1(0, 0, 80, 80);
        sf::IntRect stance2(80, 0, 80, 80);
        sf::IntRect stance3(160, 0, 80, 80);
        sf::IntRect stance4(240, 0, 80, 80);
        sf::IntRect walk1(0, 80, 80, 80);
        sf::IntRect walk2(80, 80, 80, 80);
        sf::IntRect walk3(160, 80, 80, 80);
        sf::IntRect walk4(240, 80, 80, 80);
        sf::IntRect jump2(80, 160, 80, 80);
        sf::IntRect jump3(160, 160, 80, 80);
        sf::IntRect jump4(240, 160, 80, 80);
        sf::IntRect jump5(320, 160, 80, 80);
        sf::IntRect down1(80, 480, 80, 80);
        sf::IntRect downK1(160, 480, 80, 80);
        sf::IntRect sPunch1(80, 320, 80, 80);
        sf::IntRect sPunch2(160, 320, 80, 80);
        sf::IntRect wPunch1(80, 240, 80, 80);
        sf::IntRect wPunch2(160, 240, 80, 80);
        sf::IntRect wPunch3(240, 240, 80, 80);
        sf::IntRect kick1(80, 400, 80, 80);
        sf::IntRect kick2(160, 400, 80, 80);
        thor::FrameAnimation walk;
        thor::FrameAnimation stance;
```

```
thor::FrameAnimation back;
thor::FrameAnimation jump;
thor::FrameAnimation down;
thor::FrameAnimation sPunch;
thor::FrameAnimation downK;
thor::FrameAnimation wPunch;
thor::FrameAnimation kick;
walk.addFrame(1.0f, walk1);
walk.addFrame(1.0f, walk2);
walk.addFrame(2.0f, walk3);
walk.addFrame(1.0f, walk4);
back.addFrame(1.0f, walk4);
back.addFrame(2.0f, walk3);
back.addFrame(1.0f, walk2);
back.addFrame(1.0f, walk1);
stance.addFrame(1.0f, stance1);
stance.addFrame(1.0f, stance2);
stance.addFrame(1.0f, stance3);
stance.addFrame(1.0f, stance4);
jump.addFrame(1.0f, jump2);
jump.addFrame(1.0f, jump3);
jump.addFrame(1.0f, jump4);
jump.addFrame(1.0f, jump5);
down.addFrame(1.0f, down1);
downK.addFrame(1.0f,downK1);
sPunch.addFrame(1.0f, sPunch1);
sPunch.addFrame(1.0f, sPunch2);
wPunch.addFrame(0.5f, wPunch1);
wPunch.addFrame(1.0f, wPunch2);
wPunch.addFrame(0.3f, wPunch3);
kick.addFrame(1.0f, kick1);
kick.addFrame(1.5f, kick2);
thor::Animator<sf::Sprite, std::string> animator;
animator.addAnimation("walk", walk, sf::seconds(0.4f));
animator.addAnimation("stance", stance, sf::seconds(0.4f));
animator.addAnimation("back", back, sf::seconds(0.4f));
animator.addAnimation("jump", jump, sf::seconds(0.5f));
animator.addAnimation("down", down, sf::seconds(0.4f));
animator.addAnimation("sPunch", sPunch, sf::seconds(0.3f));
animator.addAnimation("downK",downK, sf::seconds(0.1f));
animator.addAnimation("wPunch", wPunch, sf::seconds(0.2f));
animator.addAnimation("kick", kick, sf::seconds(0.4f));
bool isCrouching=false;
bool isPunching=false;
bool isDownKicking=false;
sf::RenderWindow window(sf::VideoMode(1000, 1000), "New PT3");
sf::Event event;
sf::Texture texture;
if(!texture.loadFromFile("sprite.png")){
	std::cout << "Impossible to load sprite" << std::endl;
	return 1;
}
//texture.setSmooth(true);
```

```
	sf::Clock frameClock;
	animator.playAnimation("stance", true);
	sf::Sprite sprite;
	sprite.setTexture(texture);
	sprite.setTextureRect(walk1);
	sprite.setScale(2.0f, 2.0f);
	sprite.setOrigin(position);
	window.setVerticalSyncEnabled(true);
	while(window.isOpen()){
		while(window.pollEvent(event)){
			if(event.type == sf::Event::Closed || (event.type==sf::Event::KeyPressed
&& sf::Keyboard::isKeyPressed(sf::Keyboard::Escape))){
				window.close();
			}
			//if(event.type==sf::Event::KeyPressed && event.key.code ==
sf::Keyboard::Right){
			if(sf::Keyboard::isKeyPressed(sf::Keyboard::Right)){
				sprite.move(2.0f, 0.0f);
				if(animator.getPlayingAnimation()!="walk"){
					animator.playAnimation("walk", true);
				}
			}
			if(event.type==sf::Event::KeyReleased){
				animator.playAnimation("stance", true);
				isPunching=false;
				isCrouching=false;
				isDownKicking=false;
			}
			if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left)){
				sprite.move(-2.0f, 0.0f);
				if(animator.getPlayingAnimation()!="back"){
					animator.playAnimation("back", true);
				}
			}
			if(sf::Keyboard::isKeyPressed(sf::Keyboard::Up)){
				if(animator.getPlayingAnimation()!="jump"){
					animator.playAnimation("jump", true);
				}
			}
			if(sf::Keyboard::isKeyPressed(sf::Keyboard::Down) && !isCrouching){
				if(animator.getPlayingAnimation()!="down"){
					animator.playAnimation("down", true);
					isCrouching = true;
				}
			}
			if(sf::Keyboard::isKeyPressed(sf::Keyboard::W) && isCrouching && !
isDownKicking){
				if(animator.getPlayingAnimation()!="downK"){
					animator.playAnimation("downK", false);
					isDownKicking=true;
				}
			}
```

```
                        if(sf::Keyboard::isKeyPressed(sf::Keyboard::A) && !isPunching && !
isCrouching){
                                if(animator.getPlayingAnimation()!="sPunch"){
                                        animator.playAnimation("sPunch", false);
                                        isPunching=true;
                                }
                        }
                        if(sf::Keyboard::isKeyPressed(sf::Keyboard::Z) && !isPunching && !
isCrouching){
                                if(animator.getPlayingAnimation()!="wPunch"){
                                        animator.playAnimation("wPunch", false);
                                        isPunching=true;
                                }
                        }
                        if(sf::Keyboard::isKeyPressed(sf::Keyboard::E) && !isPunching && !
isCrouching){
                                if(animator.getPlayingAnimation()!="kick"){
                                        animator.playAnimation("kick", false);
                                        isPunching=true;
                                }
                        }
                }
                if(!animator.isPlayingAnimation() && !isCrouching){
                        animator.playAnimation("stance", true);
                }
                else if(!animator.isPlayingAnimation() && isCrouching){
                        animator.playAnimation("down", true);
                }
                animator.animate(sprite);
                animator.update(frameClock.restart());
                window.clear();
                window.draw(sprite);
                window.display();
        }
        return 0;
}
```

~160 Lignes

Liens :

https://www.exobaston.com/les-10-meilleurs-beat-em-all-2019/ - recherche pour l'étude de l'existant .

http://www.capcom.com/ - étude de l'exitant : CAPCOM .

https://fr.wikipedia.org/wiki/Mortal_Kombat - étude de l'existant Mortal Kombat .

https://www.goclecd.fr/les-10-meilleurs-jeux-de-combat-de-2019-en-ce-moment/ - étude de l'existant, recherche de jeux de combat .

https://www.afjv.com/chiffres_jeux_video.php – statistiques .

https://www.lemonde.fr/pixels/article/2019/06/28/concevoir-un-jeu-video-combien-ca-coute-combien-ca-rapporte_5482520_4408996.html – prix moyen de la production d'un jeu vidéo

https://www.team-aaa.com/fr/breve/vs-fighting-2019-le-suivi-des-tournois_113534?page=2 -infos sur différent tournoi de jeu VS Fighting

https://www.fdjesport.fr/ - française des jeux, e-sport

https://www.afjv.com/news/9546_chiffre-d-affaires-du-marche-francais-du-jeu-video-en-2018.htm - 4hiffres d'affaires du jeu vidéo en France en 2018

https://www.sfml-dev.org/index-fr.php - site d'SFML

https://bromeon.ch/libraries/thor/index.html - site de THOR

https://www.piskelapp.com/ - site de Piskel

Banque d'images :

FreePik - dessin
Pixabay & Unslash - photographie