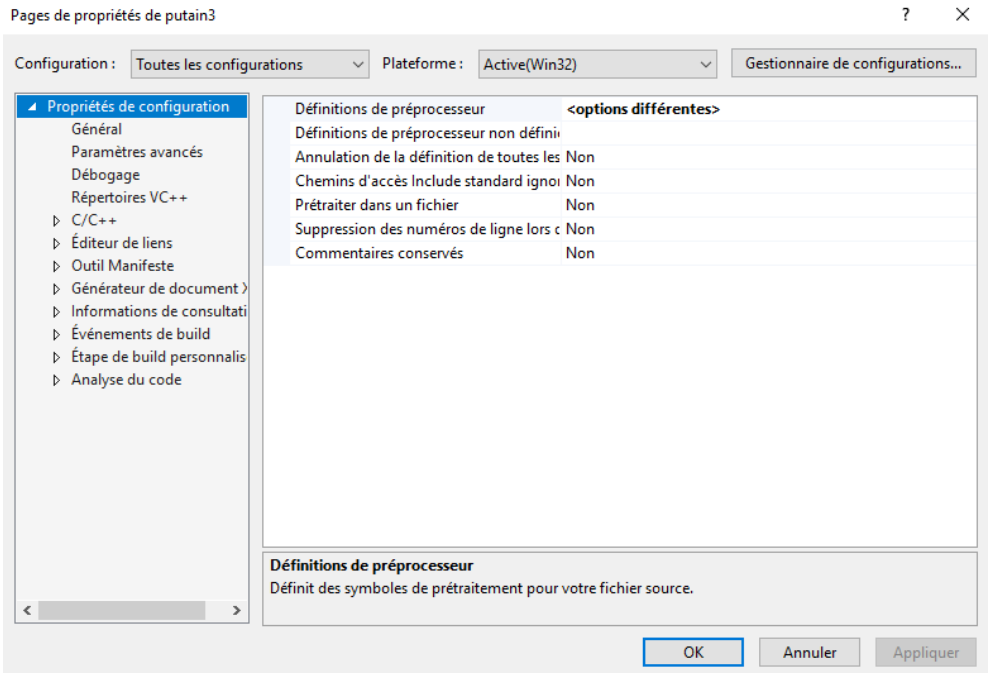
Installation SFML

Tout d’abord, il faut télécharger la dernière version de SFML ici : https://www.sfml-dev.org/download-fr.php

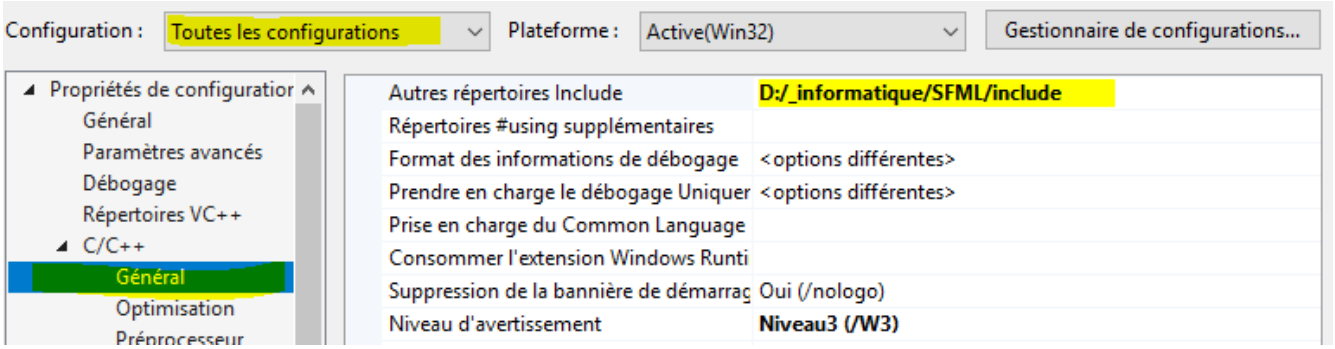
Table des matières

Installation SFML avec Visual Studio

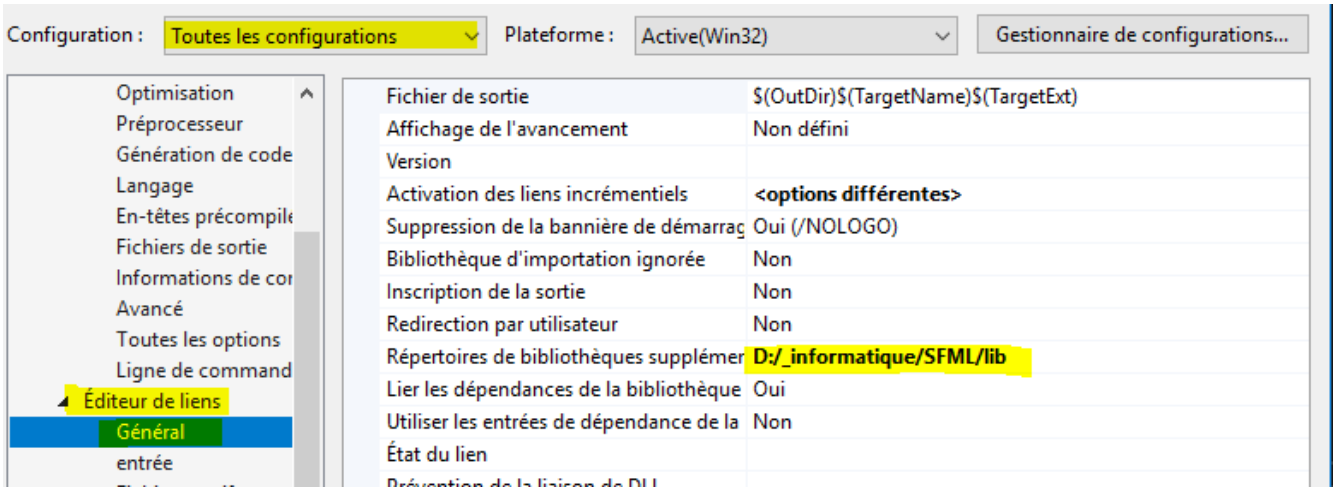
Pour Visual Studio, après avoir créer un projet on se rend dans : Projet > propriétés de ‘nom du projet’



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.

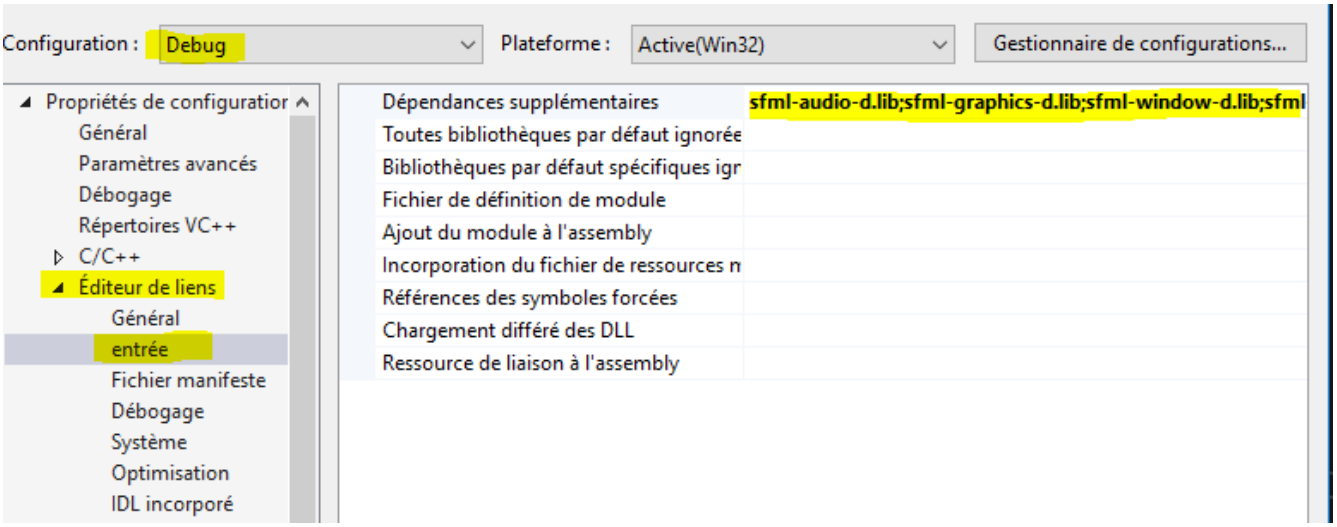


faite attention à la configuration, nous verrons qu'elle changera plus tard
Il faut aussi importer le dossier lib d'SFML, pour cela



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

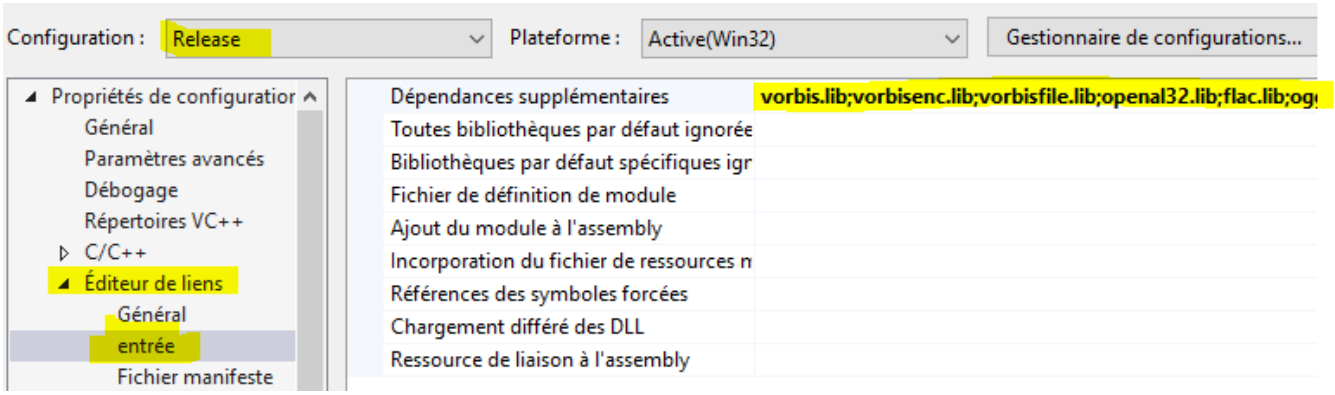


Et on y ajoute les dépendances avec la librairie SFML, pour cela , ajouter avec un copier/coller :

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 :

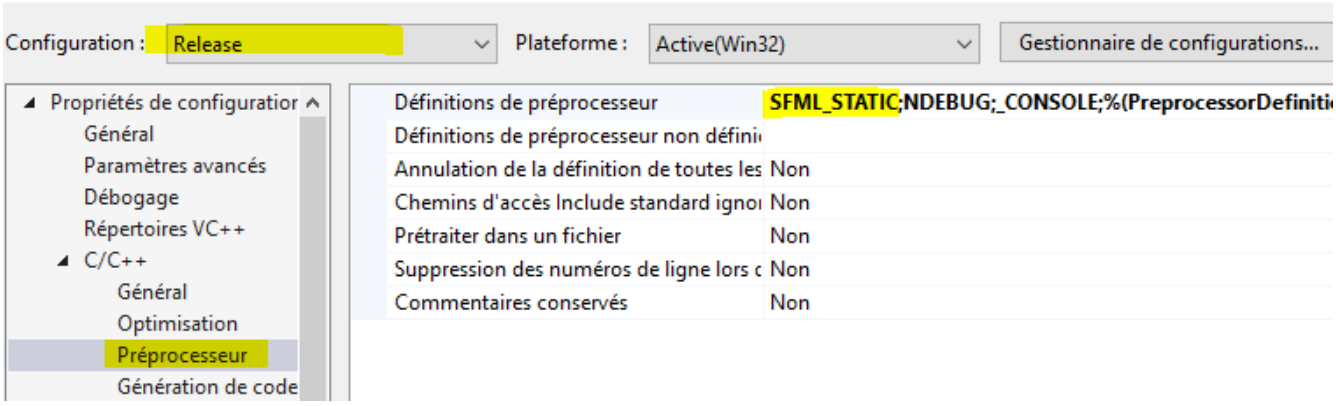


Voici les dépendances a copier :

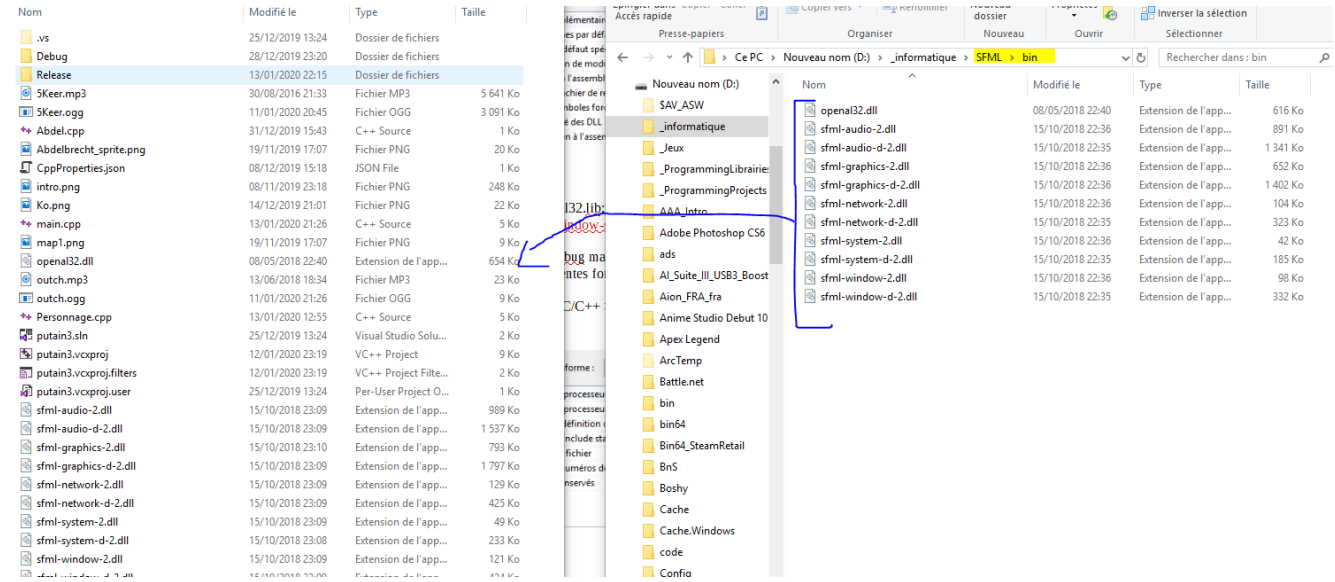
vorbis.lib;vorbisenc.lib;vorbisfile.lib;openal32.lib;flac.lib;ogg.lib;winmm.lib;opengl32.lib;freetype.lib;
sfml-audio-s.lib;sfml-graphics-s.lib;sfml-window-s.lib;sfml-system-s.lib;

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’ .

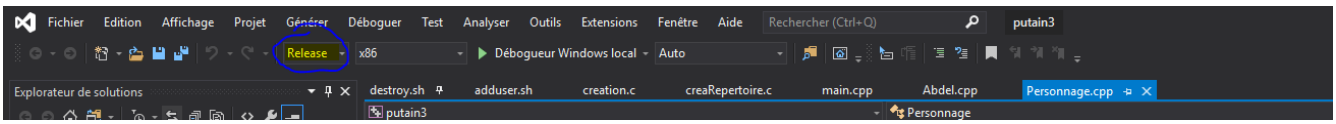


Au cas où il y aurait toujours des erreurs lors de la compilation, effectuer l'étape suivante :



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'

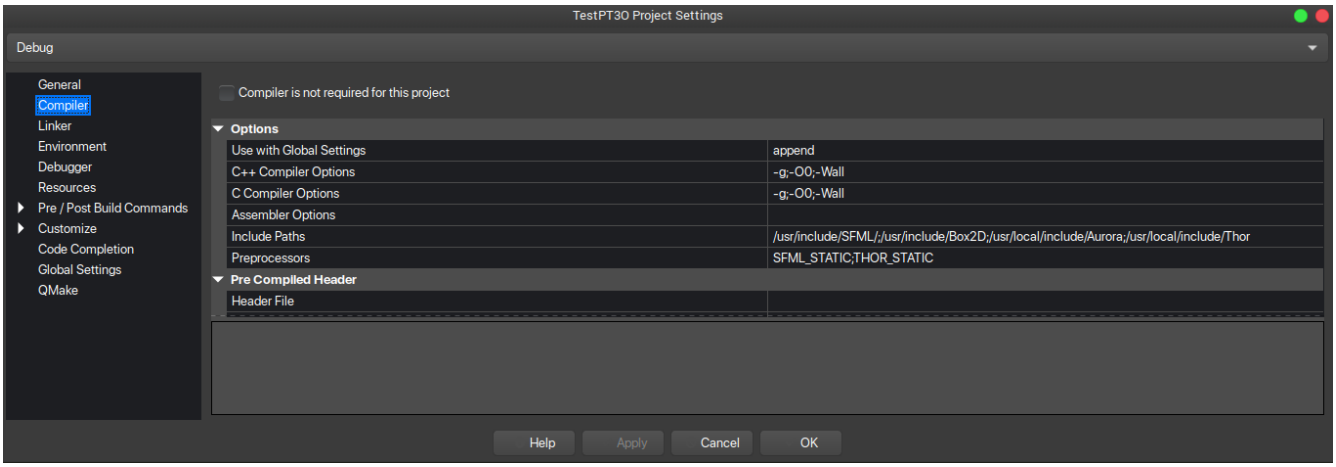


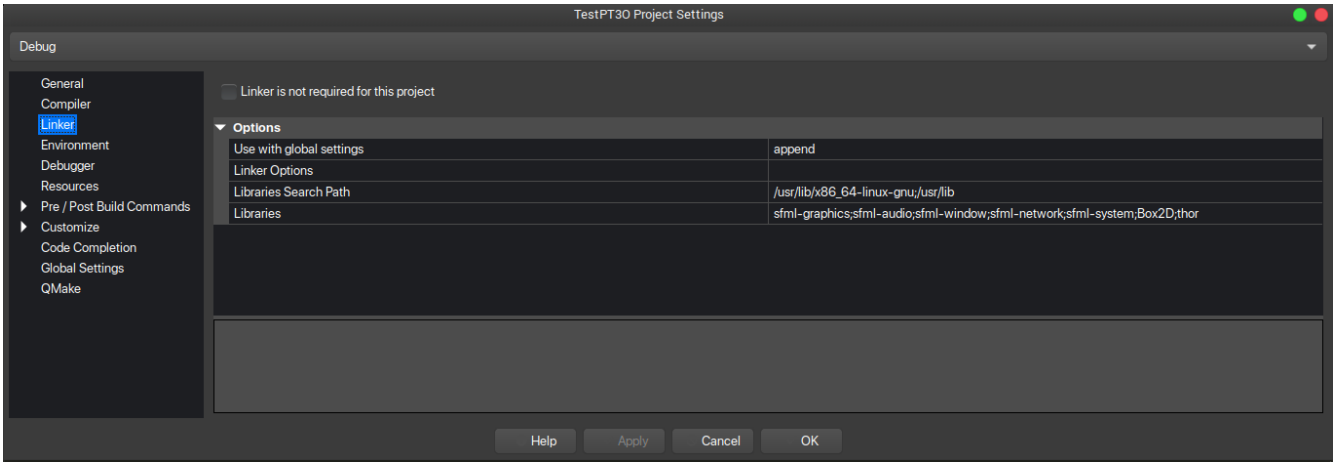
Installation SFML sur Linux

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

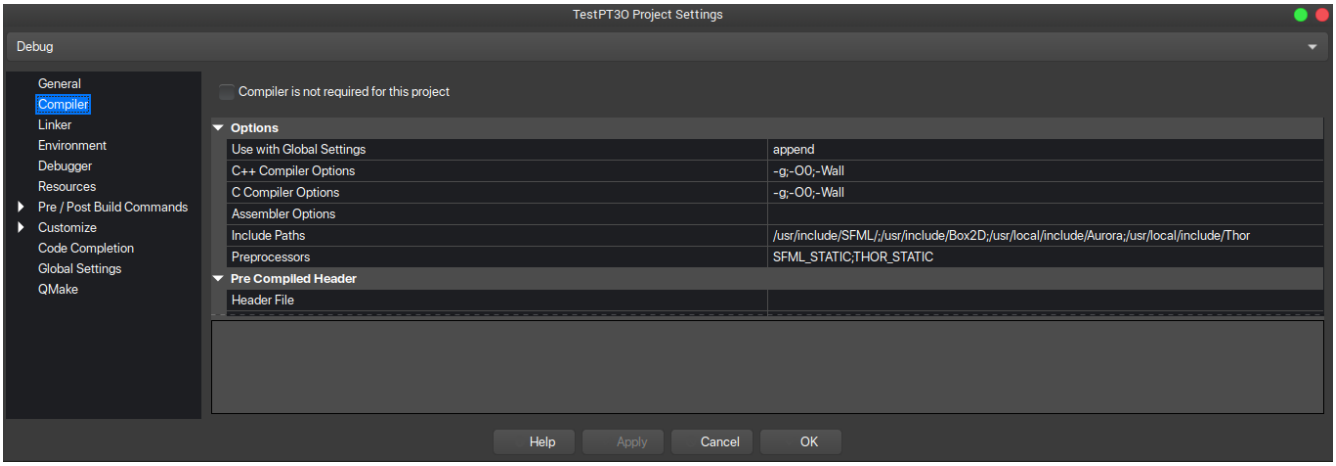
Télécharger Thor ici : https://bromeon.ch/libraries/thor/download/index.html

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/

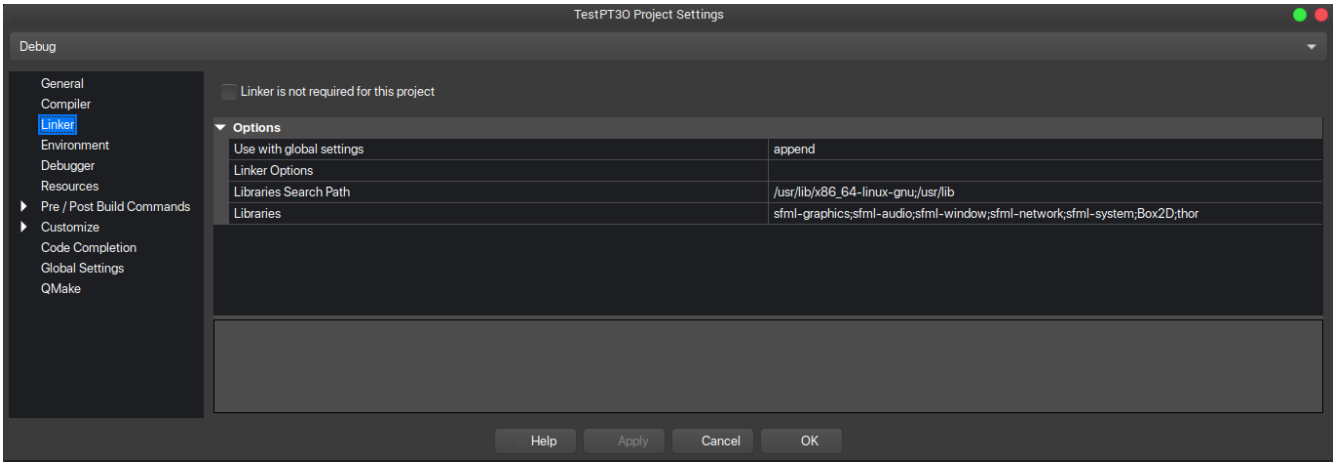
Une fois ces 2 étapes faites, Thor est correctement installé

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.



Code Principale

Le code entouré est celui effectué par Benjamin Ziehl, le reste est d'Arthur Simon

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;

windowI.setActive(false);
//lancement de l'introduction
while (windowI.isOpen())
{
	if (introTimer <= introMax) {
		windowI.clear();
		windowI.draw(intro);
		windowI.display();

		intro.setTextureRect(sf::IntRect(560 * (introTimer - 1), 0, 560 *
introTimer, 438));
		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

Mode d’emploi

Contrôles du Joueur 1 : personnage débutant à droite de l'écran

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

Bien sûr ce sont les touches à l’état final, pour le moment, dans le jeu principale seul les touche A & D et flèche gauche / flèche droite permettent de se déplacer, pour le code dans THOR :

A : coup faible | Z : coup fort | E : coup de pied | Q : gauche | flèche du bas : accroupi | D : droite | W+flèche du bas : coup de pied accroupi | flèche du haut : sauter ;