


BLOB DASH

Groupe : César Ombredane / Zinedeen Davousse / Bariteau Julien


Sommaire

Présentation du Projet BlobDash

Introduction :

Le projet BlobDash n'est pas qu'un jeu mais aussi une équipe. Elle est constituée de Zinedeen Davousse, notre main codeur, César Ombredane, notre artiste musical et pictural ainsi que moi-même, game et sound designer et manager du projet.

BlobDash est un jeu de plateforme en 2 dimensions où un personnage se déplace dans son environnement afin d'atteindre la plateforme d'arrivée et de terminer le niveau.

La charte graphique du projet est basée sur 4 couleurs, le noir, le blanc, le violet et le rose.

Règles du jeu :

Objectif : Atteindre la fin du niveau en se déplaçant dans l'environnement de jeu en esquivant les pièges qui vous feront perdre.

Espace de jeu : Le personnage se situe au milieu de l'écran, il peut avancer, reculer, sauter (vers le haut, vers l'avant ou vers l'arrière) mais aussi il peut dasher (projection vers l'avant à grande vitesse).

Commandes :

- ► : Avancer
- ◄ : Reculer
- ▲ : Sauter
- ▲+► : Sauter vers l'avant
- ▲+◄ : Sauter vers l'arrière
- M : Effectuer un Dash

Le personnage peut faire un double saut. Le saut s'effectue par la pression de la touche espace, plus celle-ci est enfoncé longtemps, plus le saut est long et inversement. Pour le deuxième saut, la durée de pression n'influence pas.
Le Dash ne peut s'effectuer qu'horizontalement et a un cooldown (temps d'attente avant la réutilisation de la capacité) de 2 sec.

Analyse du besoin et recherches

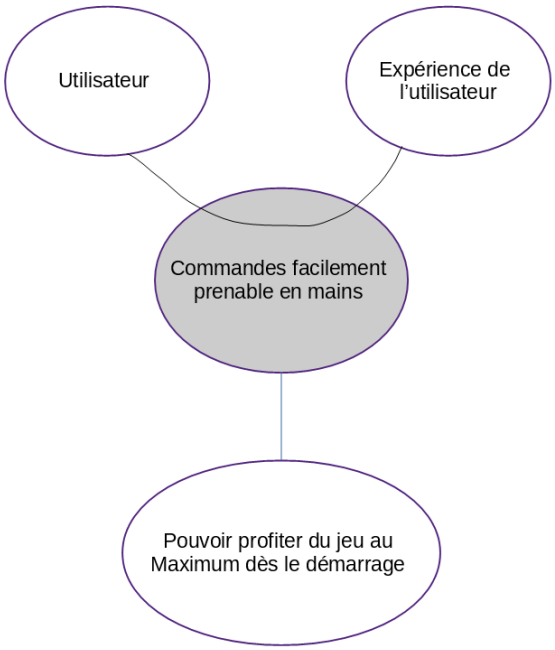
Analyse du besoin :

Le but de notre projet est de proposer à l'utilisateur un moment de plaisir mais aussi de stimulation mentale. De ce fait nous voulons créer un jeu qui reste simple dans le gameplay mais avec des mécaniques avancées.

Nous voulons toucher la plus grande plage d'utilisateur, des plus petits aux plus grands, des moins aux plus expérimentés.

Le jeu peut être classifier comme tout public car il n'y a ni violent, ni sang, ni injure. Le jeu se veut comme un mélange de Super Mario et de GeometryDash qui sont tout très populaires chez les enfants mais aussi chez les joueurs expérimentés

Les commandes restent cependant faciles à prendre en mains
, les nombres de touches est limités pour garantir une bonne expérience de jeu dès le démarrage.



Recherches :

Notre jeu est codée en C++ avec la bibliothèque SFML. Il est réalisé sur Atom et Visual Studio en tant qu'IDE.

Au début du projet nous pensions réaliser le jeu sous le moteur graphique Unity mais nous avons décidé de passer sous C++ pure avec SFML pour avoir un code authentique.

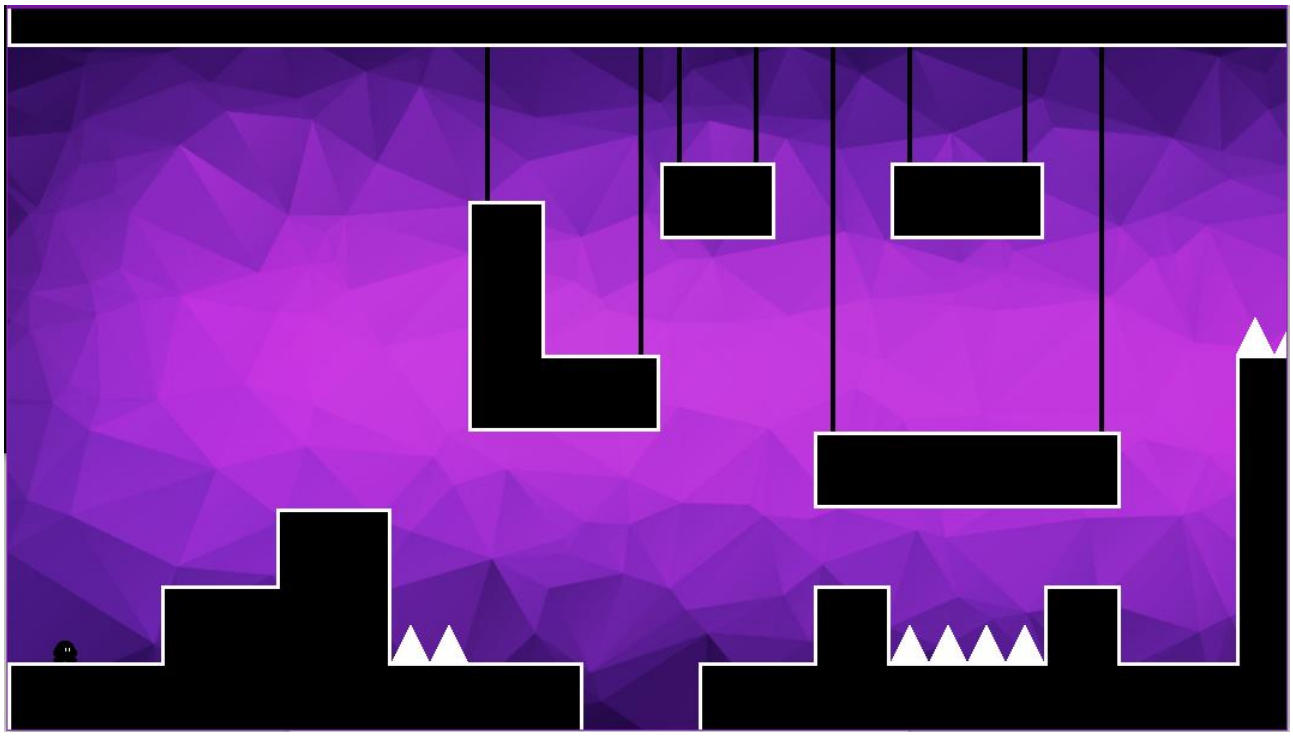
Dans notre réalisation nous nous sommes appuyés sur le tutoriel BIG TUTO de Meruvia afin de comprendre au mieux le fonctionnement de la bibliothèque.

Notre idée de jeu était aussi un jeu de plateforme mais celui devait être très nerveux avec des mouvements très rapide pour avoir un jeu « spectacle » c'est à dire un jeu plaisant a regarder pour la beauté du gameplay. Cependant notre niveau et le temps ne nous permettaient de réaliser ceci.

Notre modèle de référence était Ori and The Blind Forest où le personnage doit rejoindre la fin du niveau le plus rapidement possible.

Notre idée de départ était d'avoir un personnage de type Blob, une boule de liquide visqueuse, dans un milieu similaire mais avec des couleurs différentes (sombres/ Orange-Marron)

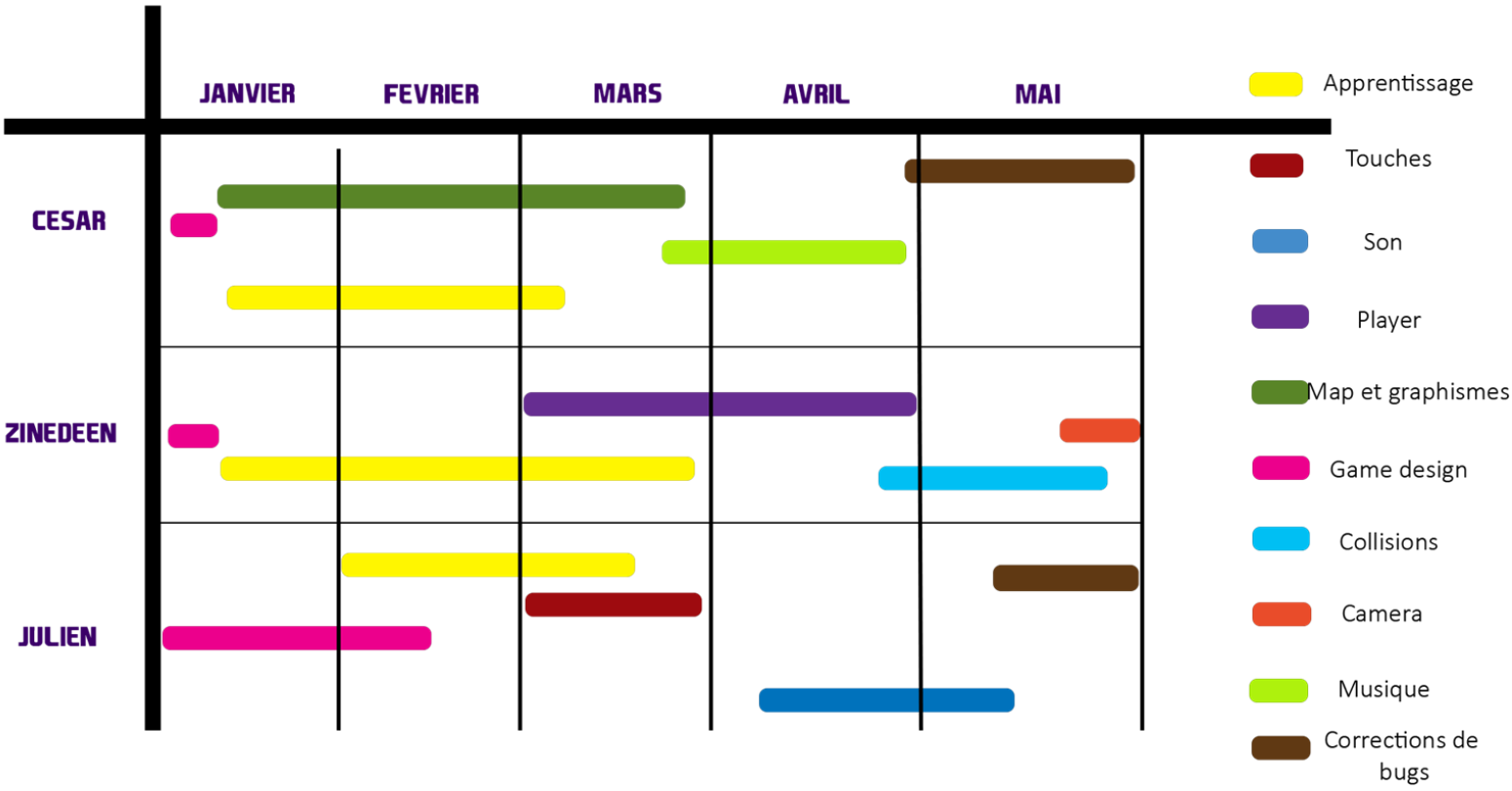
Nous avons changé de style graphique pour passer à un personnage ressemblant au personnage du jeu japonais populaire Kirby.
Le fond de notre décor est rose et violet avec un effet de lowPoly (technique utilisée pour optimiser les objets 3d dans les jeux vidéo)
Nos objets au premier plan sont noirs avec des contours blancs pour les objets sur lesquels on peut marcher dessus et en blanc les éléments qui font perdre le joueur.
Le personnage est quant à lui noir avec des yeux blancs.



L'espace de jeu

Le personnage

Répartition des taches



Nous avons pu rester en contact pour notre travail en dehors des cours grâce à la plateforme collaborative BitBucket mais aussi grâce à un service de serveurs vocaux et textuels qui se nomme Discord.

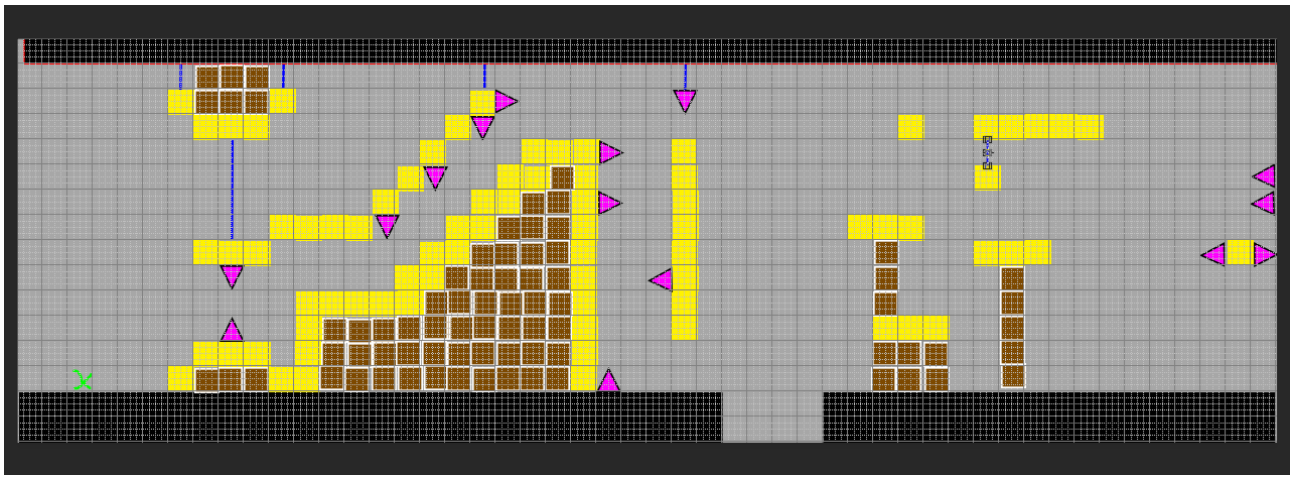
Réalisation

Game designing :

La première étape de mon travail a été de mettre en place le besoin de l'utilisateur ainsi que des règles du jeu.

Suite à cela j'ai mis en place la carte ou l'espace de jeu en délimitant l'espace de jeu pour avoir une expérience complète mais pas trop longue pour la découverte de notre jeu puis j'ai défini chaque bloc (pic, carré plein, cordes etc.) puis je les ai placés.

D'abord sur une feuille avec un papier et un crayon puis ensuite sur ordinateur avec le quadrillage adapté à la taille de l'espace de jeu en pixel, chaque bloc devait faire 8x8 pixels



Sur cette V1 de notre map, les blocs jaunes sont ceux où il y a collision, les roses sont les mortels et les marrons sont les traversables.

Sur notre version les blocs ont aussi la caractéristique de collision afin de faciliter le code

Pour que le jeu est un sens, il nous fallait un personnage, j'ai donc apporté l'idée de faire un blob car il est très simple à animer est un character design qui est libre de droit.

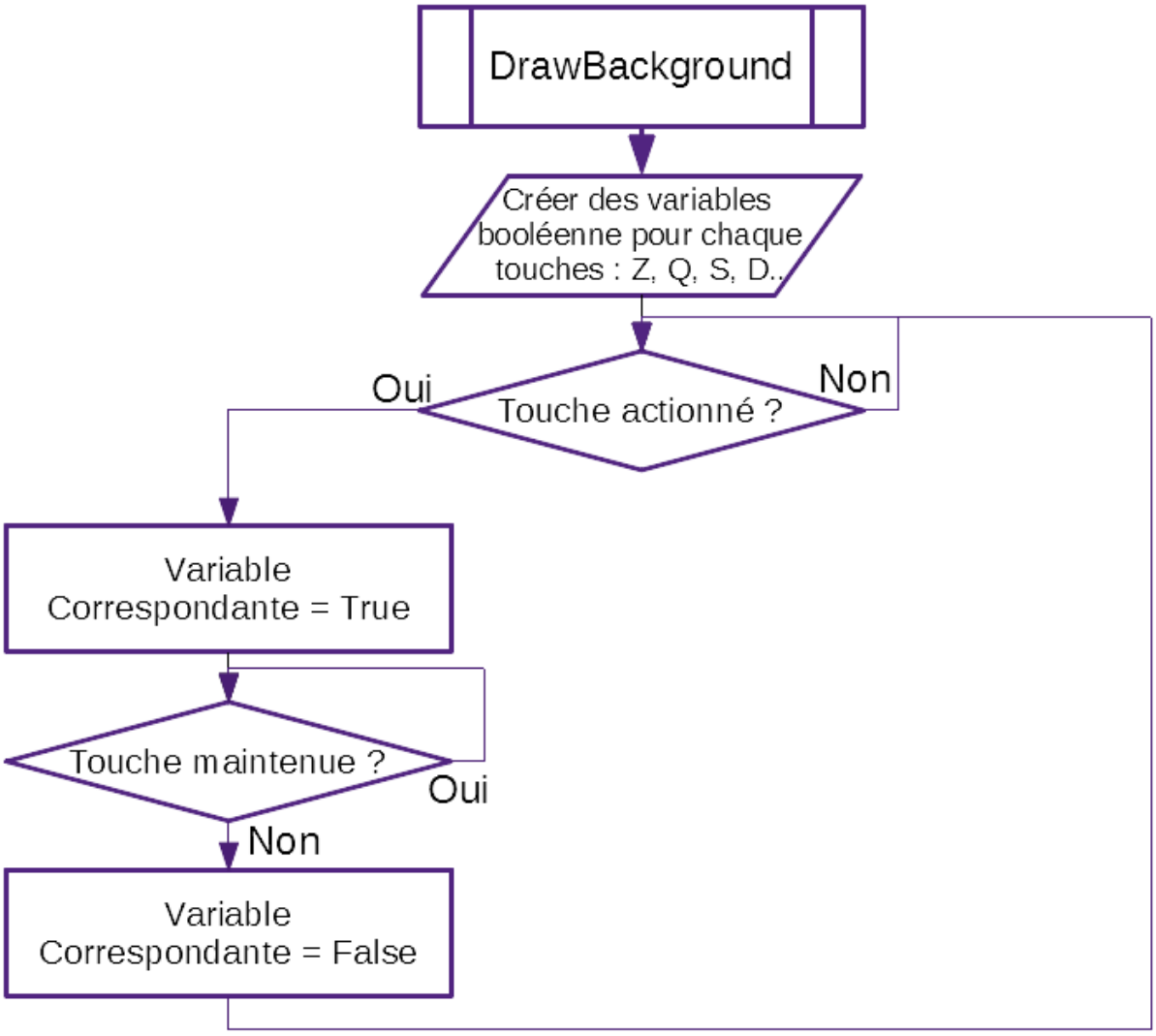
Nous sommes repartis, sur un personnage qui garde cet aspect de boule mais avec des bras et des jambes pour avoir une animation propre et compréhensible.

Inputs :

J'ai ensuite travaillé sur la gestion des inputs, ce qui est de récupérer l'information des touches du clavier.

En effet la récupération des inputs se fait par la détection de si la touche est enfoncée ou non. Si la touche reste enfoncée, l'action continue de se produire (exemple : pour marcher, on reste appuyer sur D).
Si la touche reste appuyée, la valeur booléenne reste en *True* et dès qu'elle est relâchée elle repasse en *False*, qui est son état de base.

On peut associer cette partie du code a cet algorigramme :



Algorigramme de la fonction input

Sound Designing et intégration des sons au jeu :

J'ai également fait la partie son du jeu, en termes de bruitages. Le personnage, lorsqu'il fait une action ou qu'il interagit avec son environnement (exemple : lorsqu'il rentre en collision avec un mur.)

90 % de sons ont été réalisé par moi-même, j'enregistrai plusieurs sons avec mon micro puis je les mixais, rajoutais des effets dessus pour obtenir les sons que je voulais avoir.

Les autres sont des sons pris sur des banques de sons gratuites afin d'éviter tout problèmes de droits d'auteurs. J'ai mixer ces sons et rajouté des effets sur ces sons pour obtenir des sons uniques pour notre jeu

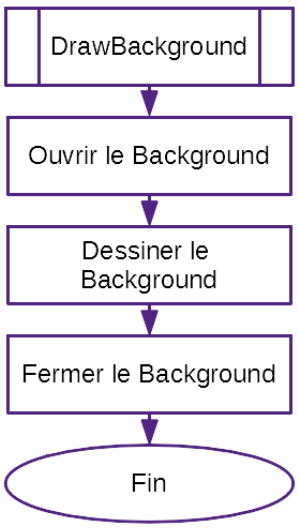
Le mixage et l'enregistrement des sons ont été effectuer sur le logiciel Audacity.

J'ai donc réalisé tous ces sons :

- Son de pas
- Son de course
- Sons d'expression du saut du personnage
- Son de collision a un mur
- Son du Dash
- Son de collision + découpage avec un pic
- Son de mort

Background :

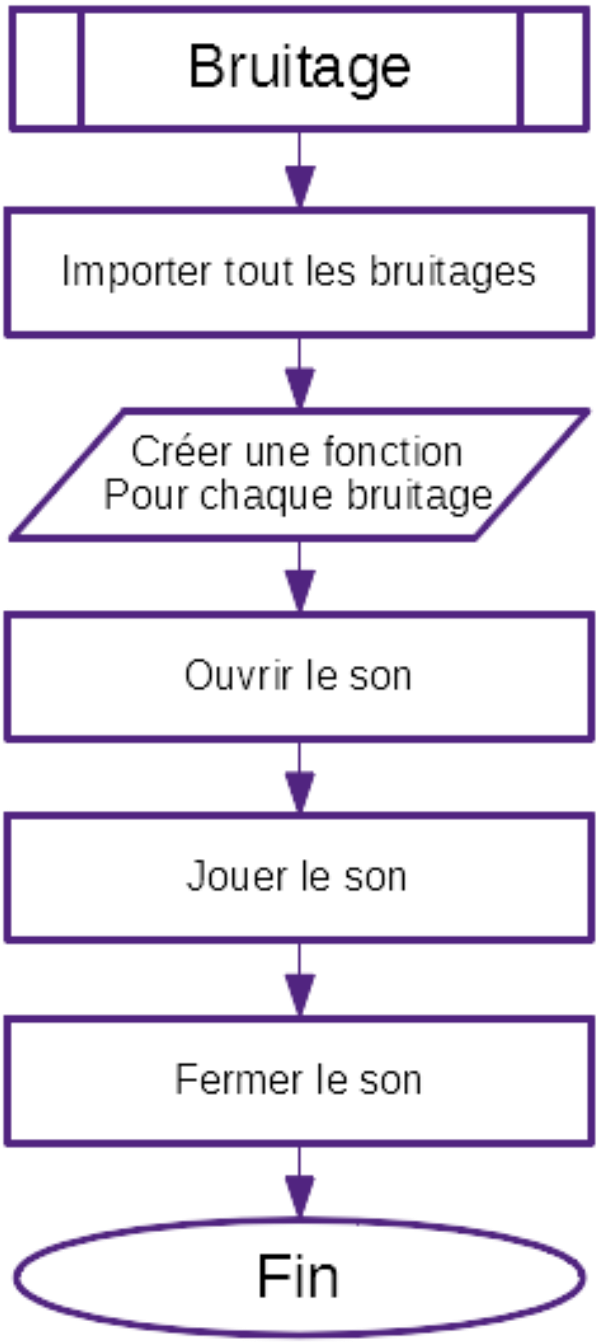
J'ai aussi participé à la réalisation et a l'intégration du background :



Background et algorigramme de la fonction DrawBackgrpound

Bruitages :

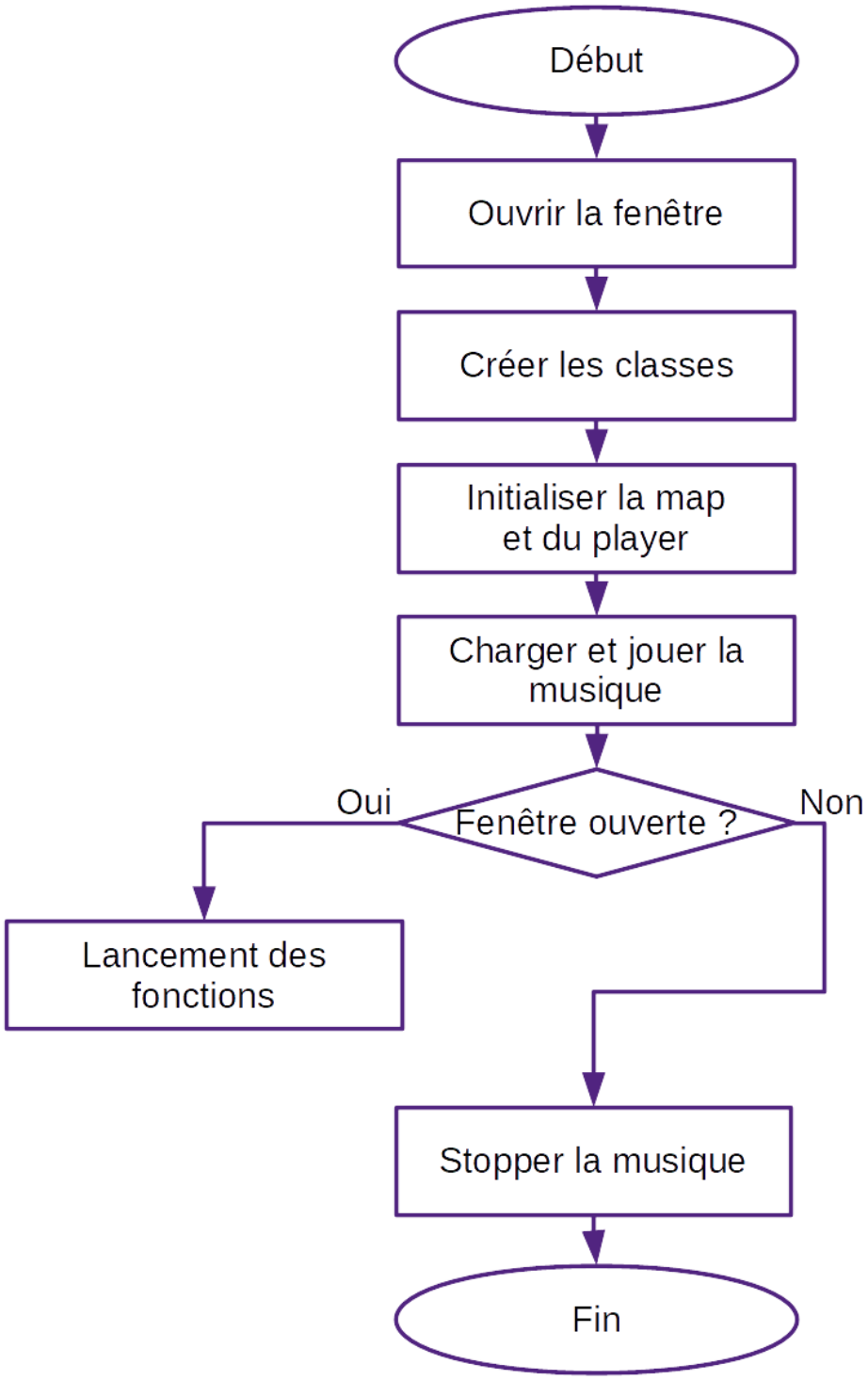
La fonction bruitage créer plusieurs fonctions qui pourrons être appeler dans le Player.
Les fonctions sont basiques mais aide grandement à la fluidité du programme.



Algorigramme de la fonction Bruitage

Main :

Bien sûr, toutes ces fonctions sont réunies dans la fonction main (fonction principal)
Elle permet de regrouper toutes les fonction e de les ordonner.



Algorigramme de la fonction Main

Intégration

Difficultés rencontrées :

La principale difficulté que j'ai rencontré durant ce projet a été la différence de niveau entre moi et mes collègues en codage.
Cependant ils ont été très compréhensif et m'ont bien accompagné dans mon travail.

Pour pallier à mon manque d'expérience on a décidé de me placer en tant que manager afin d'annoncer les objectifs et de les situer dans le temps pour le bon déroulement du projet.

Intégration :

La partie du game design est là base du jeu, sans elle le jeu n'existe pas. Tout le codage réalisé par la suite se base sur le cahier des charges et les contraintes que j'ai mis en place au début du projet. De plus, la partie graphique et audio se basent aussi sur ces conditions.

La partie des Inputs permet à Zinedeen de pouvoir mettre en mouvement le personnage après l'avoir créé dans l'espace de jeu. Sans elles la définition de « jeu » n'est pas respectée.

Les sons permettent au jeu d'avoir une dimension plus immersive. Le jeu ne reste pas plat, il se développement dans plusieurs de nos sens (la vue, le toucher, l'ouïe).

Validation :

A la fin de notre projet, nous avons fait beaucoup de tests afin de percevoir et corriger le plus de bugs possibles. Il en reste évidemment mais ils sont minimes.

Nous sommes arrivés à créer un jeu qui répond parfaitement à notre cahier des charges. Tous les objectifs sont remplis.

Cependant, le jeu n'est pas fini. En effet, il n'y a pas de menu, il n'y a qu'un seul niveau et pas de réel but (ni de monstres).

Ce seront donc des aspects du jeu à améliorer dans le futur.

Bilan et Perspectives

Notre jeu est fonctionnel, les principales fonctions sont opérationnelles, cependant il reste quelques bugs que nous n'avions pas eu le temps de corriger dû à leur complexité ou alors que leur source n'a pas été déterminée.

Malgré cela l'expérience de l'utilisateur reste limitée, il ne peut jouer qu'a un seul niveau et ne possède pas de menu pour choisir a quoi il veut jouer dans le jeu.

L'ajout de compétences et des monstres dans le jeu pourrait rajouter de la profondeur de gameplay mais aussi une hausse de la difficulté des niveaux.

Ce projet m'a permis d'améliorer mes capacités d'adaptation au rythme peut gérer des situations que techniques que je ne pouvais pas gérer tout seul.

De plus j'ai dû m'adapter à eux et modifier des règles du jeu en fonction de nos capacités de programmation.

Ce que l'ISN m'a apporté :

Je pensais que ma voie future était dans l'informatique mais je n'étais pas sûr, j'ai donc voulu m'y inscrire pour éviter de faire une erreur de parcours pour mes études supérieurs. En effet le codage m'attire vraiment très peu, ce que j'ai aimé dans ce projet c'était de travailler avec des collègues et de conduire un projet vers sa réussite.

Diffusion du Projet

Lien du Bitbucket : https://bitbucket.org/Enderguard3/isn-2.0/src/master/

Lien du Discord : https://discord.gg/uYWPns

Lien du Prezi : https://prezi.com/p/cisejb4fbwfx/#present

ANNEXES

Main.cpp

```
#include "main.h"
int main(int argc, char *argv[])
{
	RenderWindow window(VideoMode(SCREEN_WIDTH, SCREEN_HEIGHT, 32),
		"BlobDash");
	Image icon;
	if (!icon.loadFromFile("graphics/icon.png"))
		exit(EXIT_FAILURE);
	window.setIcon(16, 16, icon.getPixelsPtr());
	window.setVerticalSyncEnabled(true);
	Input input;
	Player player;
	Map map;
	Music music;
	sf::View cam;
	player.initialize(map);
	map.loadMap();
if (!music.openFromFile("sound/music.wav"))
		return -1;
	music.setVolume(5);
	music.setLoop(true);
	music.play();
	while (window.isOpen())

	{
		input.gestionInputs(window);
		window.clear();
		map.drawBackground(window);
		map.draw(window);
		player.draw(window);
		player.mapColision(map);
		player.deplacement(input, map);
		window.display();
	}
	return 0;//fin
}
```

Main.h

```
#include <cstdlib>
#include <iostream>
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include <SFML/Audio.hpp>
#include <stdafx.h>

#include "input.h"
#include "map.h"
#include "Player.h"

using namespace std;
using namespace sf;

const int SCREEN_WIDTH = 1066;
const int SCREEN_HEIGHT = 600;
```

Input.cpp

```
#include "input.h"
using namespace std;
using namespace sf;
Input::Input()
{
        button.left = button.right = button.down = button.jump =
        button.dash = false;
}
Input::Button Input::getButton(void) const
{
     return button;
}
void Input::setButton(int bouton, bool etat)
{
    switch (bouton)
    {
    case down:
        button.down = etat;
        break;
    case right:
        button.right = etat;
        break;
    case left:
        button.left = etat;
        break;
    case dash:
        button.dash = etat;
        break;

    case jump:
        button.jump = etat;
        break;
    case escape:
            button.escape = etat;
            break;
    }
}
void Input::gestionInputs(RenderWindow &window)
{
    getInput(window);
}
void Input::getInput(RenderWindow &window)
{
      while (window.pollEvent(event))
    {
        switch (event.type)
        {
        case Event::Closed:
            window.close();
            break;
        case Event::KeyPressed:
            switch (event.key.code)
```

```
            {
            case Keyboard::Escape:
                window.close();
                break;
            case Keyboard::Space:
                button.jump = true;
                break;


            case Keyboard::M:
                button.dash = true;
                break;
            case Keyboard::Q:
                button.left = true;
                break;
            case Keyboard::D:
                button.right = true;
                break;
            case Keyboard::S:
                button.down = true;
                break;
            default:
                break;
            }
            break;
        case Event::KeyReleased:
            switch (event.key.code)
            {
            case Keyboard::Space:
                button.jump = false;
                break;
            case Keyboard::Q:
                button.left = false;
                break;
            case Keyboard::D:
                button.right = false;
                break;
            case Keyboard::S:
                button.down = false;
                break;

            default:
                break;
            }
            break;

        default:
            break;
        }
    }
}
```

Input.h

```
#ifndef INPUT_H
#define INPUT_H
#include <SFML/Graphics.hpp>
class Input
{
    struct Button { bool left, right, escape, down, jump, dash; };
public:
    Input();
    Button getButton(void) const;
    void setButton(int bouton, bool etat);
    void gestionInputs(sf::RenderWindow &window);
    void getInput(sf::RenderWindow &window);
private:
    sf::Event event;
    Button button;
    const enum {down, right, left, dash, jump, escape};
};
#endif
```

Map.cpp

```
#include "map.h"
using namespace std;
using namespace sf;
Map::Map()
{
	if (!backgroundTexture.loadFromFile("graphics/background.png"))
	{
cout << "Erreur durant le chargement de l'image de background." << endl;
	}
	else
		background.setTexture(backgroundTexture);
	if (!tileSetTexture.loadFromFile("graphics/tileset.png"))
	{
cout << "Erreur durant le chargement de l'image du tileset." << endl;
	}
	else
		tileSet.setTexture(tileSetTexture);
	startX = startY = 0;
}
int Map::getBeginX(void) const { return beginx; }
int Map::getBeginY(void) const { return beginy; }
int Map::getStartX(void) const { return startX; }
int Map::getStartY(void) const { return startY; }
int Map::getMaxX(void) const { return maxX; }
int Map::getMaxY(void) const { return maxY; }
int Map::getTile(int x, int y) const { return tile[y][x]; }
void Map::setStartX(int valeur) { startX = valeur; }
void Map::setStartY(int valeur) { startY = valeur; }
void Map::drawBackground(RenderWindow &window{
	window.draw(background);
}
void Map::loadMap()
{
	fstream fin;
	int x = 0;
	int y = 0;
	maxX = 0;
	maxY = 0;
	vector < vector < int > > lignes;
	vector < int > myVectData;
	string strligne, strcara;
	stringstream iostr;
	fin.open("map/map.txt");
	if (!fin.is_open())    {
		cerr << "Erreur de chargement du fichier.";
	}
	while (!fin.eof())
	{
		getline(fin, strligne);
		if (!strligne.size())
			continue;
		iostr.clear();
```

```
            iostr.str(strligne);
            myVectData.clear();
            while (true)
            {
                  getline(iostr, strcara, ' ');
                  myVectData.push_back(atoi(strcara.c_str()));
                  if (!iostr.good()) break;
            }

            if (myVectData.size())
                  lignes.push_back(myVectData);
      }
      fin.close();
      beginx = lignes[0][0];
      beginy = lignes[0][1];
      for (x = 2; x < MAX_MAP_X + 2; x++)
      {
            tile[y][x - 2] = lignes[y][x];
      }
      for (y = 1; y < MAX_MAP_Y; y++)
      {
            for (x = 0; x < MAX_MAP_X; x++)
            {
                  tile[y][x] = lignes[y][x];
                  if (tile[y][x] > 0)
                  {
                        if (x > maxX)
                        {
                              maxX = x;
                        }
                        if (y > maxY)
                        {
                              maxY = y;
                        }
                  }
            }
      }
      maxX = (maxX + 1) * TILE_SIZE;
      maxY = (maxY + 1) * TILE_SIZE;
}

void Map::draw(RenderWindow &window)
{
      int x, y, mapX, x1, x2, mapY, y1, y2, xsource, ysource, a;
      mapX = startX / TILE_SIZE;
      x1 = (startX % TILE_SIZE) * -1;
      x2 = x1 + SCREEN_WIDTH + (x1 == 0 ? 0 : TILE_SIZE);
      mapY = startY / TILE_SIZE; //pareil pour mapY
      y1 = (startY % TILE_SIZE) * -1;
      y2 = y1 + SCREEN_HEIGHT + (y1 == 0 ? 0 : TILE_SIZE);
      for (y = y1; y < y2; y += TILE_SIZE)
      {
            mapX = startX / TILE_SIZE;
            for (x = x1; x < x2; x += TILE_SIZE)
            {
```

```
                  a = tile[mapY][mapX];
                  ysource = a / 10 * TILE_SIZE;
                  xsource = a % 10 * TILE_SIZE;
                  tileSet.setPosition(Vector2f(x, y));
                  tileSet.setTextureRect(sf::IntRect(xsource, ysource,
TILE_SIZE, TILE_SIZE));
                  window.draw(tileSet);
                  mapX++; //incrementation de mapX
            }
            mapY++; //incrementation de mapY
      }
}
void Map::testDefilement(void)
{
      if (startX < maxX - SCREEN_WIDTH - 1)
            startX += 1;
}
```

Map.h

```
#ifndef MAP_H
#define MAP_H
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <vector>
class Map
{
public:
      Map();
      int getBeginX(void) const;
      int getBeginY(void) const;
      int getStartX(void) const;
      int getStartY(void) const;
      int getMaxX(void) const;
      int getMaxY(void) const;
      int getTile(int x, int y) const;
      void setStartX(int valeur);
      void setStartY(int valeur);
      void drawBackground(sf::RenderWindow &window);
      void loadMap(); //charge la map depuis map.txt
      void draw(sf::RenderWindow &window);
      void testDefilement(void);
private:


      int beginx, beginy;
      int startX, startY;
      int maxX, maxY;
      int tile[19][50];
      sf::Texture backgroundTexture;
      sf::Sprite background;
      sf::Texture tileSetTexture;
      sf::Sprite tileSet;
      const int SCREEN_WIDTH = 1066;
      const int SCREEN_HEIGHT = 600;
      const int MAX_MAP_X = 50;
      const int MAX_MAP_Y = 19;
      const int TILE_SIZE = 32;
};
#endif
```

Player.cpp

```
#include "Player.h"
#include "main.h"
using namespace std;
using namespace sf;
Player::Player()
{
	if (!texturePlayer.loadFromFile("graphics/spritesheet.png"))
	{
cout << "Erreur durant le chargement du spritesheet du Player." << endl;
	}
	else
	spritePlayer.setTexture(texturePlayer);
	playerPosX = 0;
	playerPosY = 0;
	saveX = 0;
	saveY = 0;
	direction = 0;
	etat = 0;
	frameMax = frameTimer = frameNumber = 0;
	h = w = 0;
	xSprite = 0;
	ySprite = 0;
	isGround = false;
	isCeiling = false;
	isJumping = false;
	jumpTimer = MAX_JUMP;
	isDJumping = false;
	dJumpTimer = MAX_D_JUMP;
	isDead = false;
	b = 0;
};
void Player::initialize(Map &map)
{
	etat = IDLE;
	direction = RIGHT;
	timeBetween2Frames = IDLE_SPEED;
	frameNumber = 0;
	frameTimer = timeBetween2Frames;
	frameMax = 4;
	h = PLAYER_HEIGHT;
	w = PLAYER_WIDTH;
	playerPosX = map.getBeginX();
	playerPosY = map.getBeginY();
	spritePlayer.setPosition(50, 500);
};
void Player::deplacement(Input &input, Map &map)
{
	if (isDead)
	{
		etat = IDLE;
		direction = RIGHT;
		timeBetween2Frames = IDLE_SPEED;
		frameMax = 4;
```

```
                spritePlayer.setPosition(50, 500);
                isDead = false;
                isJumping = false;
                isDJumping = false;
                isGround = false;
                isCeiling = false;
                isWallL = false;
                isWallR = false;
        }
        if (isGround)
        {
              dJumpTimer = MAX_D_JUMP;
              isDJumping = false;
              jumpTimer == MAX_JUMP;
        }
        if (b <= 0)
        {
              if (input.getButton().dash)
              {
                    if (direction == RIGHT)
                    {
                    }
              }
        }
        if (!isGround)
        {
              etat = FALL;
              frameMax = 4;
              timeBetween2Frames = FALL_SPEED;
              spritePlayer.move(Vector2f(0,GRAVITE));
              if (input.getButton().left)
              {
                    direction = LEFT;
                    spritePlayer.move(Vector2f(-PLAYER_SPEED, 0));
              }
              if (input.getButton().right)
              {
                    direction = RIGHT;
                    spritePlayer.move(Vector2f(PLAYER_SPEED, 0));
              }
              If (jumpTimer == 0)
              {
                    if (input.getButton().jump)
                    {
                          if (dJumpTimer <= 0)
                          {
                                isDJumping = false;
                          }
                          else
                          {
                                dJumpTimer--;
                                isDJumping = true;
                          }
                    }
              }
```

```
	}
	else if (input.getButton().left)
	{
		etat = WALK;
		frameMax = 10;
		timeBetween2Frames = WALK_SPEED;
		direction = LEFT;
		spritePlayer.move(Vector2f(-PLAYER_SPEED,0));
		if (input.getButton().right)
		{
			spritePlayer.move(Vector2f(PLAYER_SPEED,0));
			direction = RIGHT;
		}
	}
	else if (input.getButton().right)
	{
		etat = WALK;
		frameMax = 10;
		timeBetween2Frames = WALK_SPEED;
		direction = RIGHT;
		spritePlayer.move(Vector2f(PLAYER_SPEED,0));
		if (input.getButton().left)
		{
			spritePlayer.move(Vector2f(-PLAYER_SPEED,0));
			direction = LEFT;
		}
	}
	else
	{
		etat = IDLE;
		frameMax = 4;
		timeBetween2Frames = IDLE_SPEED;
	}
	if (isCeiling)
	{
		isJumping = false;
		jumpTimer = 0;
		if (isDJumping)
		{
			isDJumping = false;
			dJumpTimer = 0;
		}
	}
	if (isWallR)
		spritePlayer.move(Vector2f(-PLAYER_SPEED - 2, 0));
	if (isWallL)
		spritePlayer.move(Vector2f(PLAYER_SPEED + 2, 0));
	if (isJumping)
	{
		spritePlayer.move(Vector2f(0,-JUMP_HEIGHT));
		etat = JUMP;
		frameMax = 6;
		timeBetween2Frames = JUMP_SPEED;
		if (input.getButton().left)
		{
```

```
                direction = LEFT;
                spritePlayer.move(Vector2f(-PLAYER_SPEED, 0));
            }
            if (input.getButton().right)
            {
                direction = RIGHT;
                spritePlayer.move(Vector2f(PLAYER_SPEED, 0));
            }
        }
        if (isDJumping)
        {
            spritePlayer.move(Vector2f(0,-JUMP_HEIGHT));
            etat = D_JUMP;
            frameMax = 8;
            timeBetween2Frames = D_JUMP_SPEED;
            if (input.getButton().left)
            {
                direction = LEFT;
                spritePlayer.move(Vector2f(-PLAYER_SPEED, 0));
            }
            if (input.getButton().right)
            {
                direction = RIGHT;
                spritePlayer.move(Vector2f(PLAYER_SPEED, 0));
            }
        }
        if (input.getButton().jump)
        {
            if (jumpTimer <= 0)
            {
                isJumping = false;
                if (isGround)
                {
                    jumpTimer = MAX_JUMP;
                }
            }
            else
            {
                jumpTimer--;
                isJumping = true;
            }
        }
        else if (!input.getButton().jump)
        {
            isJumping = false;
            jumpTimer = 0;
            if (isGround)
            {
                jumpTimer = MAX_JUMP;
            }
        }
};
int Player::getPosX(void) const { return spritePlayer.getPosition().x; };
int Player::getPosY(void) const { return spritePlayer.getPosition().y; };
void Player::draw(RenderWindow &window)
```

```
{
	if (frameTimer <= 0)
	{
		frameTimer = timeBetween2Frames;
		frameNumber++;
		if (frameNumber >= frameMax)
		frameNumber = 0;
	}
	else
	frameTimer--;
	ySprite = etat*h;
	xSprite = frameNumber*w;
	if (direction == LEFT)
	{
	spritePlayer.setTextureRect(IntRect(xSprite + 32, ySprite, -w, h));
		window.draw(spritePlayer);
	}
	else if (direction == RIGHT)
	{
		spritePlayer.setTextureRect(IntRect(xSprite, ySprite, w, h));
		window.draw(spritePlayer);
	}
};
void Player::mapColision(Map &map)
{
	spriteX = spritePlayer.getPosition().x;
	spriteY = spritePlayer.getPosition().y;
	for (i = 0; i <= 32; i++)
	{
	if (map.getTile((spriteX + 37 - i) / 32, (spriteY + 33) / 32) > 1)
			isGround = true;
else if (map.getTile((spriteX - 12 + i) / 32, (spriteY + 33) / 32) > 1)
			isGround = true;
		else
			isGround = false;
	if (map.getTile((spriteX + 37 - i) / 32, (spriteY + 12) / 32) > 1)
			isCeiling = true;
else if (map.getTile((spriteX - 10 + i) / 32, (spriteY + 12) / 32) > 1)
			isCeiling = true;
		else
			isCeiling = false;
	if (map.getTile((spriteX + 25) / 32, (spriteY + i - 1) / 32) > 1)
			isWallR = true;
		else
			isWallR = false;
	if (map.getTile((spriteX + 2.5) / 32, (spriteY + i - 1) / 32) > 1)
			isWallL = true;
		else
			isWallL = false;
		if (map.getTile((spriteX + 32 - i) / 32, (spriteY + 33) / 32) == 13
|| map.getTile((spriteX - i) / 32, (spriteY + 33) / 32) == 18 )
			isDead = true;
		if (map.getTile((spriteX + 25) / 32, (spriteY + i - 1) / 32) == 13
|| map.getTile((spriteX + 25) / 32, (spriteY + i - 1) / 32) == 18)
			isDead = true;
```

```
            if (map.getTile((spriteX + 2.5) / 32, (spriteY + i - 1) / 32) == 13
|| map.getTile((spriteX + 2.5) / 32, (spriteY + i - 1) / 32) == 18)
                    isDead = true;
      }
}
```

Player.h

```
#ifndef PLAYER_H
#define PLAYER_H
#include "input.h"
#include "map.h"
class Player
{
public:
      Player();
      void initialize(Map &map);
      void draw(sf::RenderWindow &window);
      void deplacement(Input &input, Map &map);
      void mapColision(Map &map);
      int getPosX(void) const;
      int getPosY(void) const;
private:
      int playerPosX;
      int playerPosY;
      int saveX, saveY;
      int direction;
      int etat;
bool isGround, isWallR, isWallL, isCeiling, isJumping, isDJumping, isDead;
      int frameMax;
      int frameNumber;
      int frameTimer;
      int h, w;
      int xSprite, ySprite;
      int timeBetween2Frames;
      int i;
      int spriteX;
      int spriteY;
      int jumpTimer;
      int dJumpTimer;
      int b;
      sf::Texture texturePlayer;
      sf::Sprite spritePlayer;
      const int PLAYER_WIDTH = 32;
      const int PLAYER_HEIGHT = 32;
      const float PLAYER_SPEED = 1.5;
      const int IDLE = 0;
      const int IDLE_SPEED = 15;
      const int WALK = 1;
      const int WALK_SPEED = 3;
      const int RUN = 2;
      const int RUN_SPEED = 6;
      const int JUMP = 3;
      const int JUMP_SPEED = 6;
      const int D_JUMP = 4;
      const int D_JUMP_SPEED =0;
      const int DASH = 5;
      const int DASH_SPEED = 10;
      const int WALL = 6;
      const int WALL_SPEED =0;
      const int DEAD = 7;
```

```
      const int DEAD_SPEED =0;
      const int FALL = 8;
      const int FALL_SPEED = 20;
      const int GRAVITE = 2;
      const int JUMP_HEIGHT = 5;
      const int MAX_JUMP = 30;
      const int MAX_D_JUMP = 15;
      const int DASH_LENGHT = 4;
      const int DASH_TIMER = 30;
      const int RIGHT = 1;
      const int LEFT = 2;
};
#end
```