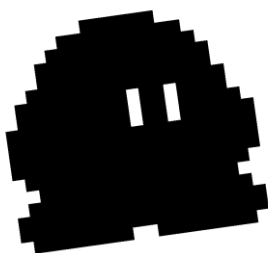
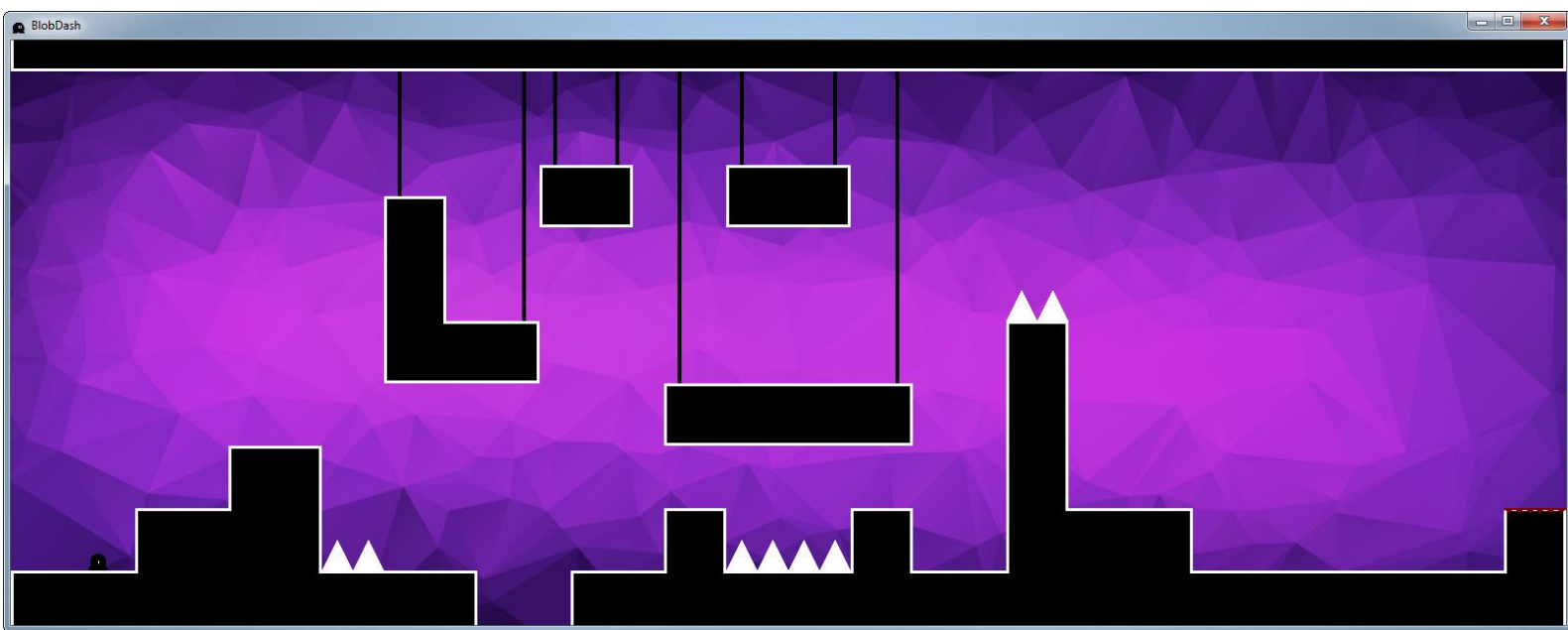


Dossier ISN

Jeu de plateforme : BlobDash



Equipe de création :

- Julien BARITEAU
- Zinédeen MOUHAMED DAVOUSSE
- César OMBREDANE

Sommaire

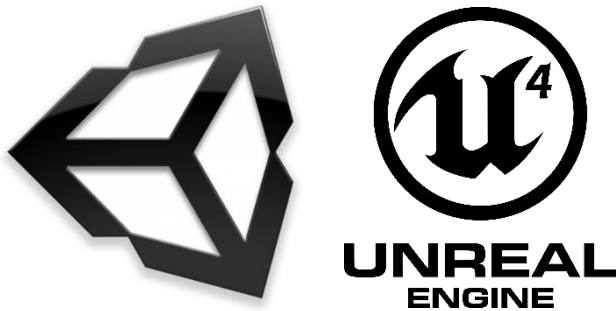
- Présentation du projet « BLOBDASH » P.3
- Cahier des charges P.4
- Analyse du besoin et recherches d'idées P.5
- Equipe/répartition des tâches P.7
- Réalisation P.8
- Intégration P.13
- Bilan et perspectives P.14
- Diffusion du projet P.14
- Annexes P.15

Présentation du projet

Mon équipe et moi sommes passionnés de jeu vidéo et, étant donné la liberté de ces projets, nous avons tout de suite pensé à en créer un. De plus, deux d'entre nous avons déjà expérimenté la programmation. Nous nous sommes donc lancés dans un projet ambitieux qui est de créer un jeu vidéo sans moteur de jeu.

Qu'est-ce qu'un moteur de jeu ?

Un moteur de jeu est un logiciel qui permet de créer un jeu vidéo sans avoir à programmer toutes les bases (il est quand même indispensable de programmer). Les plus connus sont Unity3D et UnrealEngine4.



Créer un jeu sans moteur de jeu est donc un réel défi car tous les paramètres doivent être pris en compte, même les actions les plus basiques. Car nous partons réellement de zéro et sans aucune aide.

Qu'est-ce qu'un « Platformer »

Un jeu de plateforme ou « Platformer » est un jeu vidéo généralement en 2D qui a pour but de se déplacer de plateforme en plateforme pour arriver à la fin du niveau. Au fur et à mesure des niveaux, la difficulté des sauts augmente et des ennemis de plus en plus nombreux apparaissent. Le représentant le plus connu de ce type de jeu est évidemment « Mario ».



Cahier des charges

Création du jeu vidéo :

- création des graphismes (map et personnage) * *map = carte (environnement dans lequel se déplace le personnage)*
- création de la musique et des SFX * *SFX = Sound Design Effect (bruitages du jeu)*
- programmation de l'affichage de la map
- programmation du personnage (déplacement, gravité et collision avec la map)
- intégration des annexes (musique, SFX et icône)

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 enfoncée longtemps, plus le saut est long (dans la durée) et inversement.

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

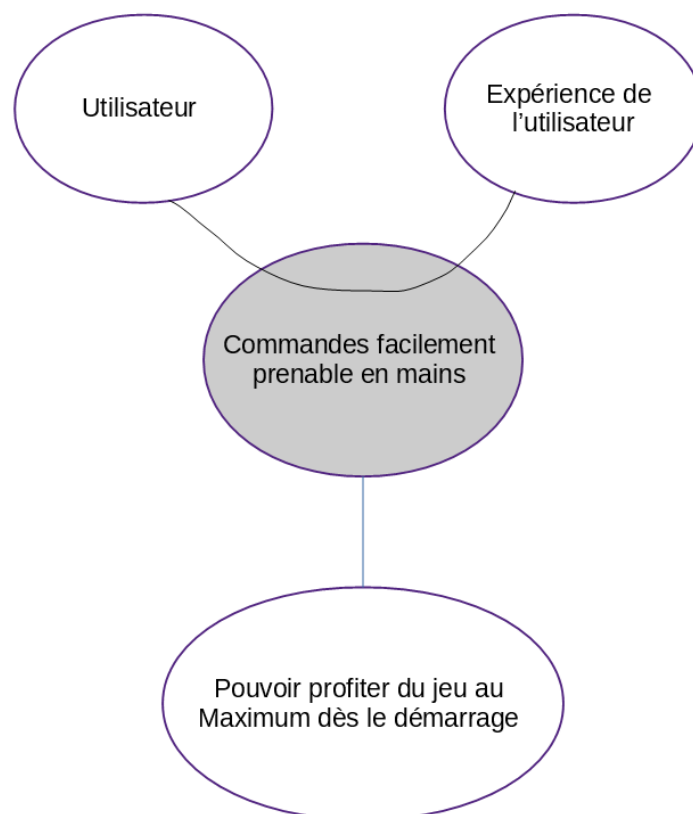
Analyse du besoin :

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 néophytes aux plus expérimentés.

Le jeu peut être classifié comme « tout public » car il n'y a ni violence, ni sang, ni injure. Le jeu se veut comme un mélange de *SuperMario* 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, le nombre de touches est limité pour garantir une bonne expérience de jeu dès le démarrage.



Recherches :

Notre jeu est codé 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 un jeu de plateforme mais celui-ci devait être très nerveux, avec des mouvements très rapides pour avoir un jeu « spectacle », c'est à dire un jeu plaisant à regarder pour la beauté du *gameplay*. Cependant notre niveau en programmation et le temps de mis en œuvre ne nous permettaient pas la réalisation souhaitée.

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 à *Ori* 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 *Low-Poly* (technique utilisée pour optimiser les objets 3D dans les jeux vidéo). Nos objets au premier plan sont noirs avec des contours blancs. Le personnage est quant à lui noir avec des yeux blancs.



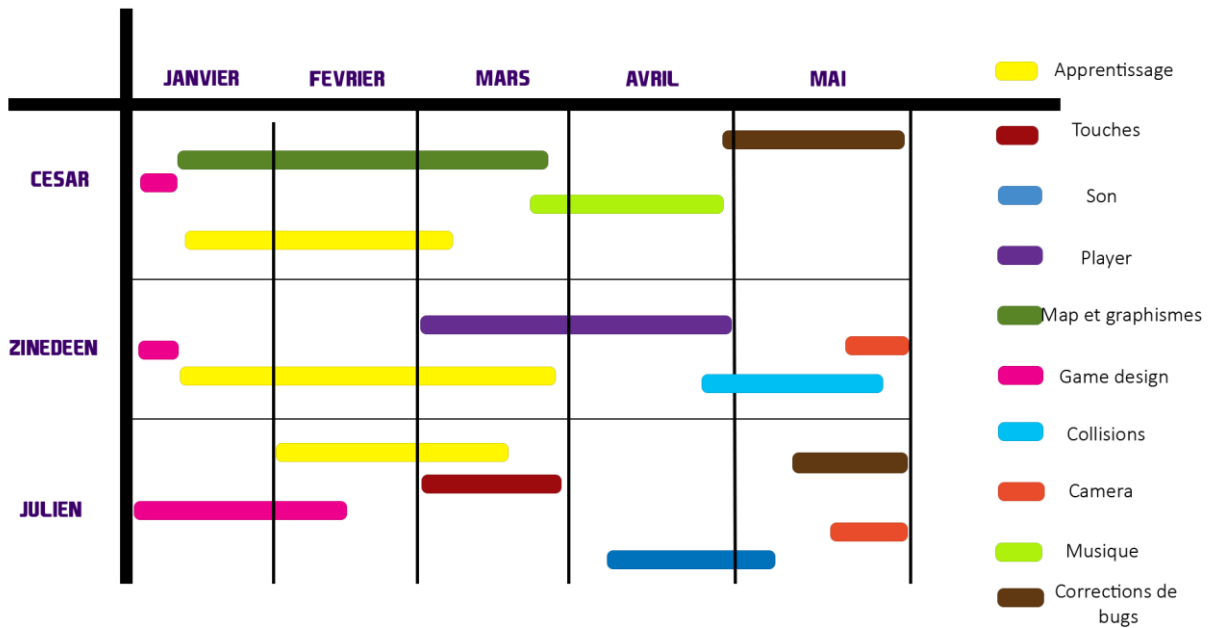
Pour les graphismes, nous avons utilisé Photoshop, Illustrator et Paint.net.

Pour l'animation, nous avons utilisé Adobe Animation.



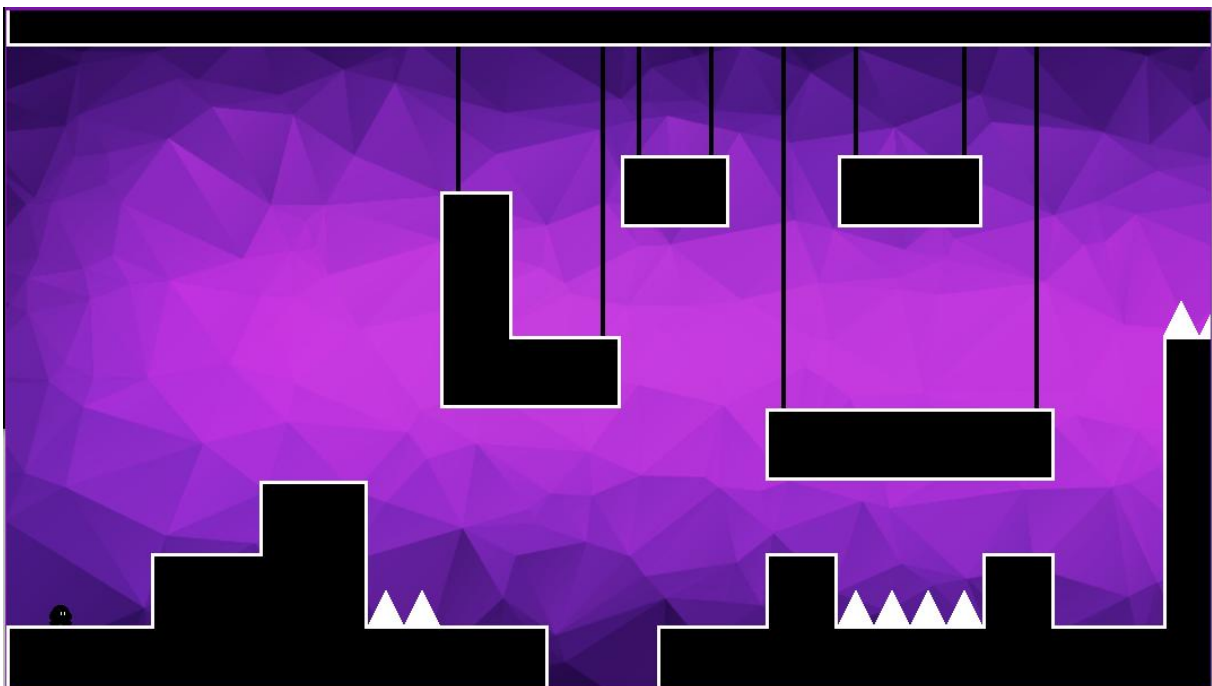
Et Pour la musique et le son, nous utilisons LogicProX et Audacity.

Répartition des tâches



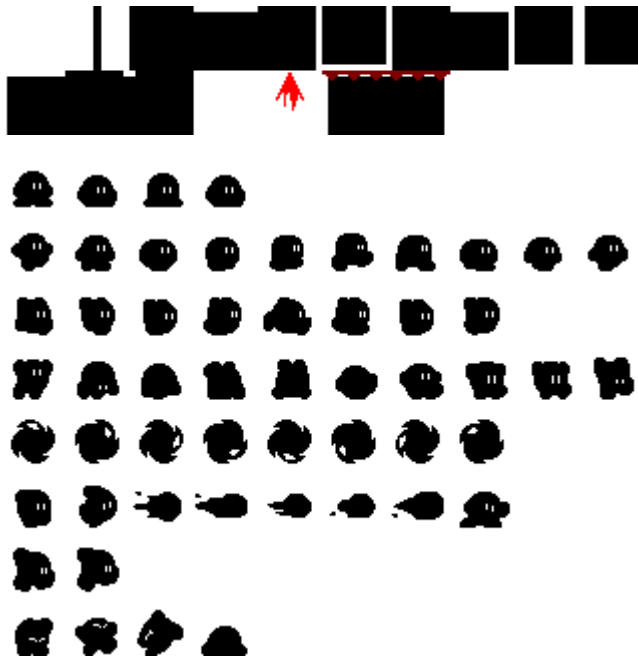
Nous avons pu rester en contact pour notre travail en dehors des cours grâce à la plateforme collaborative *BitBucket* mais aussi grâce à un service de serveurs vocaux et textuels qui se nomme *Discord*.

Aperçu du jeu



Réalisation

J'ai d'abord réalisé les graphismes du jeu :



La *tileset* est une image regroupant toutes les textures de la map qui seront ensuite organisées par le programme.

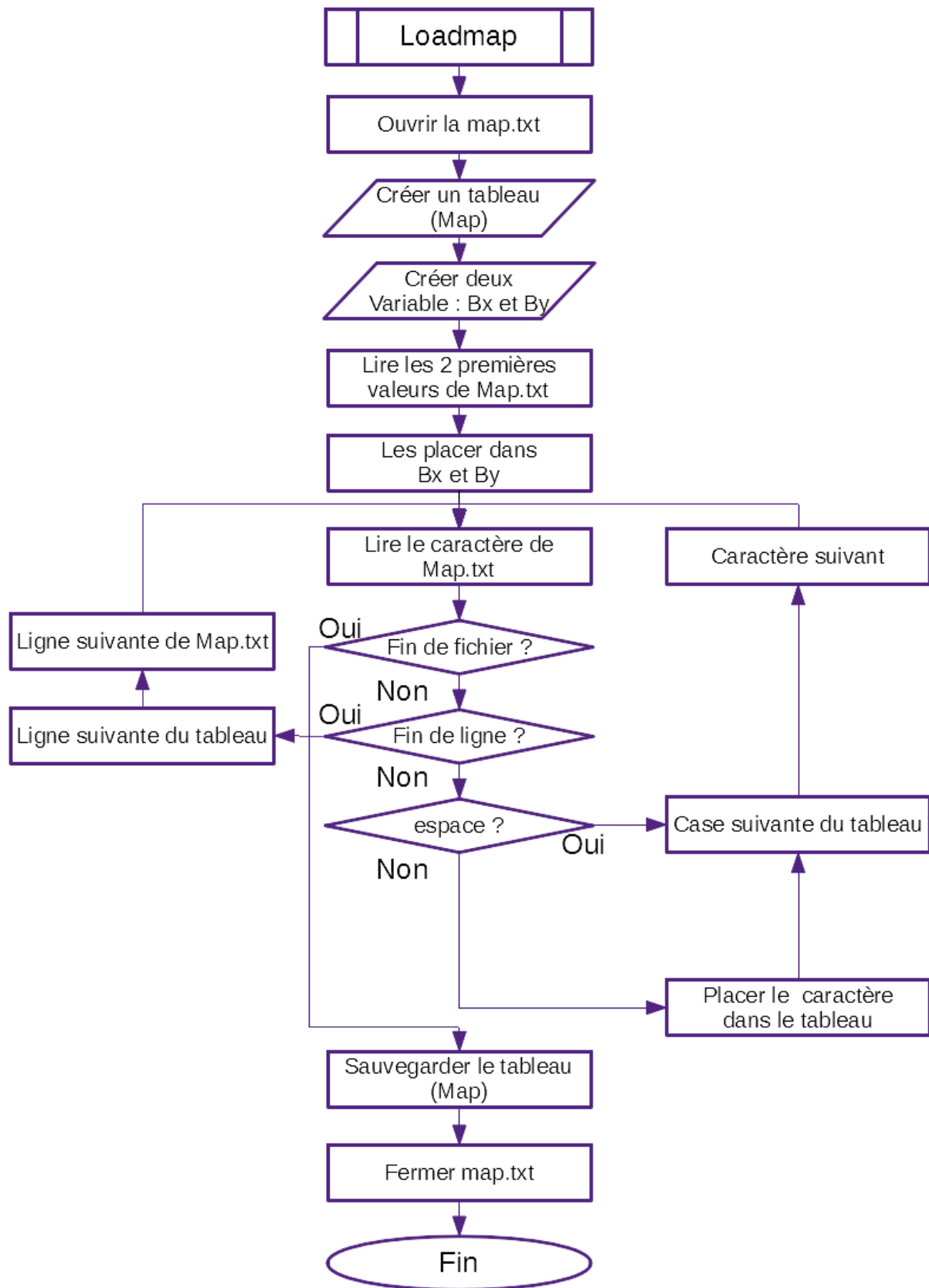
La *Spriteset* est une image regroupant tous les états du personnage. Mises à la suite, ces images forment une animation qui se joue selon les actions du joueur.



Le *Background* est aussi important car il définit l'ambiance du jeu. On y retrouve notre chartre graphique (violet et rose) et la technique *LowPoly*.

Je me suis ensuite attelé à la programmation de la map.

La map est une matrice de nombres qui représentent chacun une partie de la *tileset*. Grâce à la fonction *LoadMap*, le programme ordonne ces chiffres dans un tableau.



Algorithme de la fonction LoadMap

César OMBREDANE

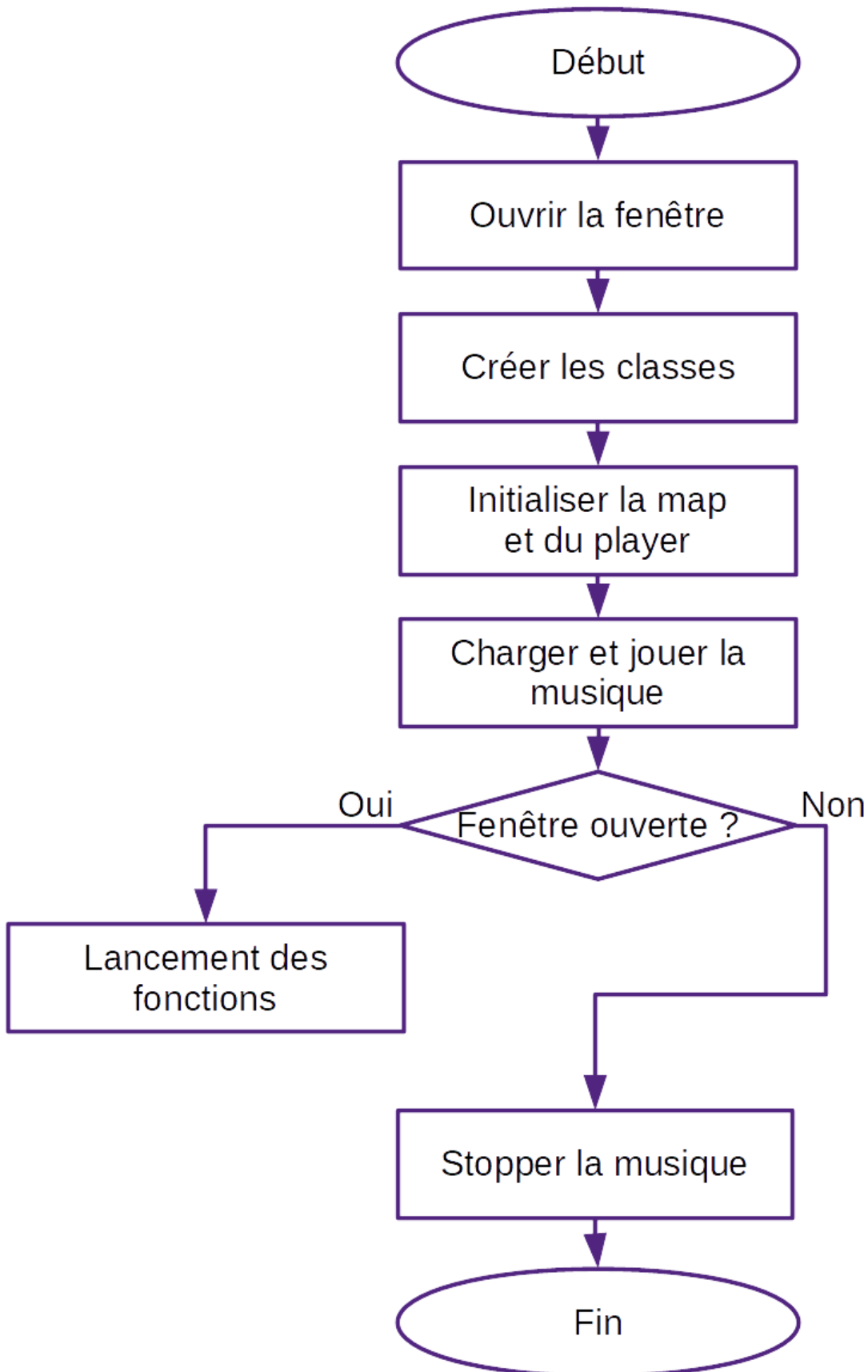
```
3254495555555555555555555555555555555555555555555555555555555558  
170000000000001000110101010010100000000000000000000000017  
170000000000001000110101010010100000000000000000000000017  
170000000000001000110101010010100000000000000000000000017  
170000000000001000110370101033701000000000000000000000017  
170000000000001070019580109558010000000000000000000000017  
17000000000000640010000100000010000000000000000000000017  
17000000000000640010000100000010000000000000000000000017  
170000000000006400100001000000100013130000000000000000017  
170000000000006113370000100000010001070000000000000000017  
17000000000000955580000100000010006400000000000000000017  
1700000000000000000000000010333333700064000000000000000017  
170000000000000000000000009555555800064000000000000000017  
17000000010370000000000000000000000000000640000000000000017  
17000000062400000000000000000000000000000640000000000000017  
1700010331224000000000000010700001070006113337000000000001516  
1700062222413130000000000641313131364000622224000000000064  
1033312222211333370001033121133331211333122222113333333333124  
62222222222222418181862222222222222222222222222222222222
```

Matrice de nombres de la map

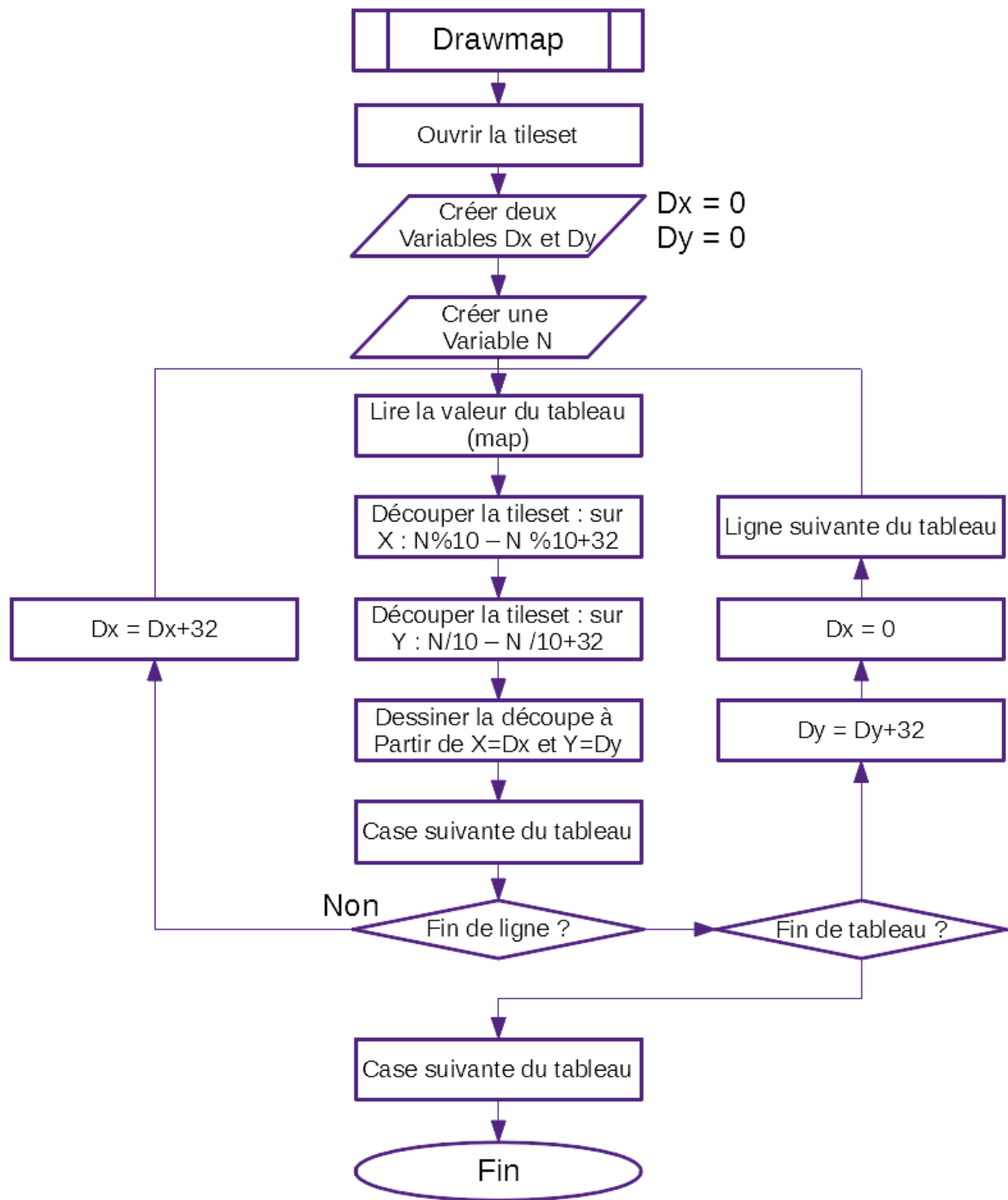
Il faut ensuite dessiner la map.

Pour cela la fonction *DrawMap* scanne chaque valeur du tableau, découpe la partie de la *tileset* associée et la dessine à un endroit définit. La map se dessine de gauche à droite et de haut en bas. Les *tiles* étant très légères (32 pixels par 32 pixels) la map se dessine très rapidement et est en permanence actualisée.

Toutes ces fonctions sont regroupées dans la fonction *Main* (fonction principale) qui lie le travail de toute l'équipe.



Algorithme de la fonction Main



Algorithme de la fonction `DrawMap`

Ma dernière tâche a été de composer la musique du jeu.

Je l'ai réalisé avec LogicProX (logiciel professionnel). La musique est une boucle de 30 secondes qui reflète l'ambiance du jeu.

Intégration

Difficultés rencontrées :

Lors de ce projet, nous nous sommes heurtés à de nombreux problèmes comme, premièrement, le langage. Nous avons dû, avant de commencer le projet, apprendre à programmer en C++. Cela nous a pris un certain temps. Nous avons ensuite dû apprendre la Library SFML.

Nous avons aussi eu beaucoup de mal, à la fin du projet, à corriger tous les « bugs » qui étaient inhérents aux techniques que nous utilisions.

Intégration :

Ma partie est la base du jeu. Rien ne pouvait se faire sans la map. C'est pour cela que ma partie a dû être réalisée avant la partie du *Player*. Dans la partie *Player*, le code se sert des fonctions et des variables de ma partie. Elle est en lien direct avec la création des collisions.

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 terminé. 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 dus à leur complexité ou alors à leur source qui n'a pas été déterminée.

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

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

Ce projet m'a permis de m'améliorer dans ma compréhension de la programmation et m'a mis face à la difficulté.

J'ai pu apprendre un nouveau langage et une nouvelle façon de programmer (le POO).

Ce que l'ISN m'a apporté :

L'ISN m'a permis pour la première fois de travailler pendant longtemps sur un projet concret qui n'était pas guidé. En effet, la plupart des projets que j'avais entrepris jusque-là (en informatique) était plus ou moins guidée. Cela m'a donc beaucoup épanoui de pouvoir tester et découvrir des choses sans chemin préconçu.

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_LENGTH = 4;
    const int DASH_TIMER = 30;
    const int RIGHT = 1;
    const int LEFT = 2;
};
#endif
```