Université Abdelmalek Essaâdi
Faculté des Sciences et Technique
de Tanger

RAPPORT DE PROJET

Réalisé par :

- Mohamed SERBOUT
- Basma EL BARKI
- Groupe : 2

Encadré par :

Pr. Ikram Ben abdel ouahab

Objective

Vieux générale de projet

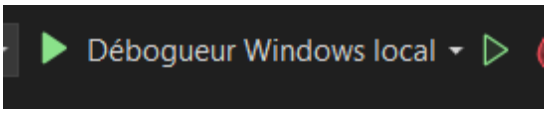
Dans ce projet on développer un jeu connue PICO_PARK à partir de logiciel sfml. L'objectif principal de ce projet est de maitriser la programmation orientée objet par la mise en place D'un jeu vidéo 2D, le jeu roller PICO_PARK, c'est un jeu qui a connu un grand succès dans les plateformes mobile.

INTRODUCTION

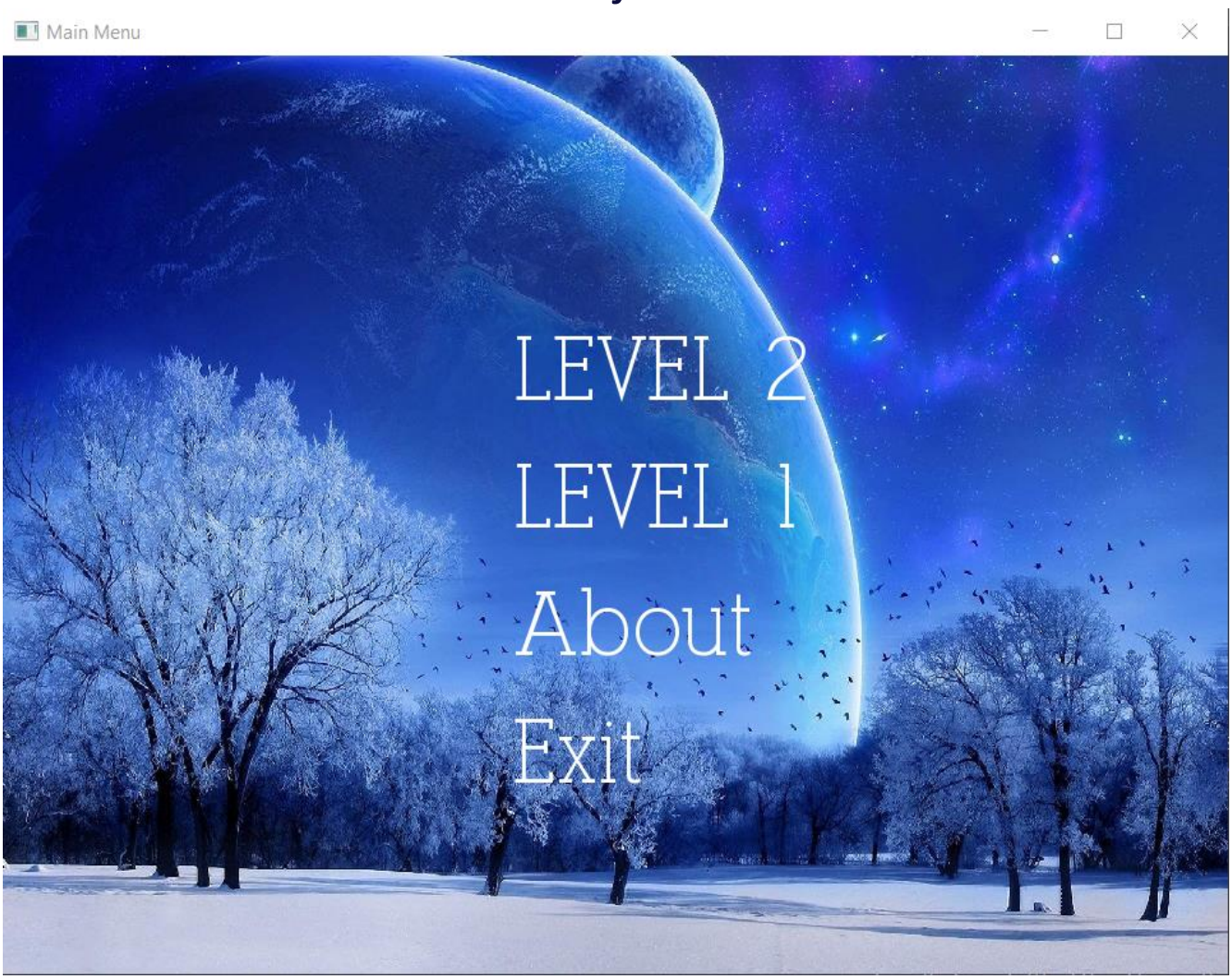
SFML est une interface de programmation destinée à construire des jeux vidéo ou des programmes interactifs. Elle est écrite en C++, mais également disponible dans divers langages comme C,D,Python,Ruby,OCaml ou Microsoft .NET. Elle a entre autres pour but de proposer une alternative orientée objet à la SDL.

Comment Jouer ?

Pour jouer double clic sur le dossier pico_park après sur fichier comporte le picoPark.sln . alors maintenant click sur Déboguer Windows local :

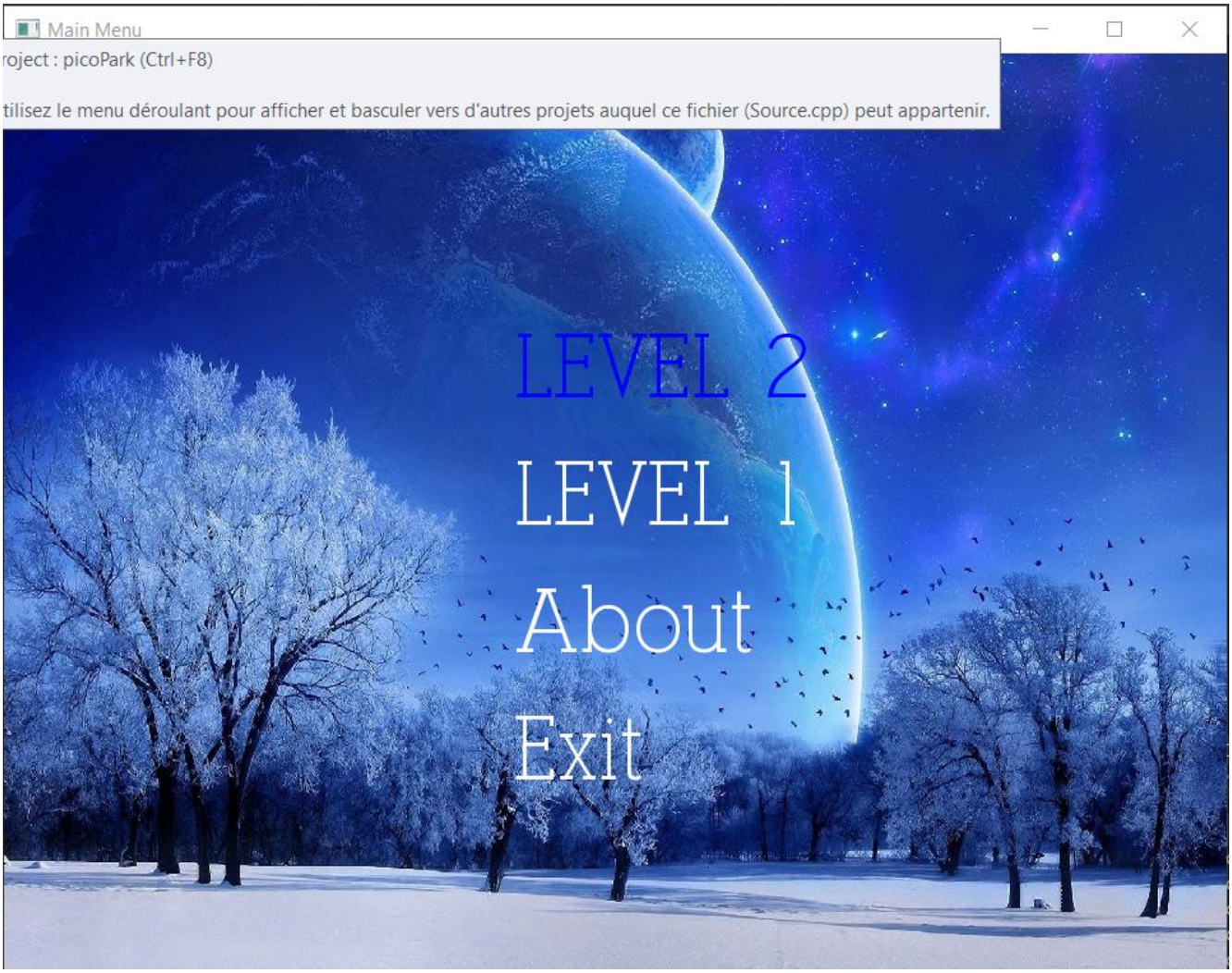


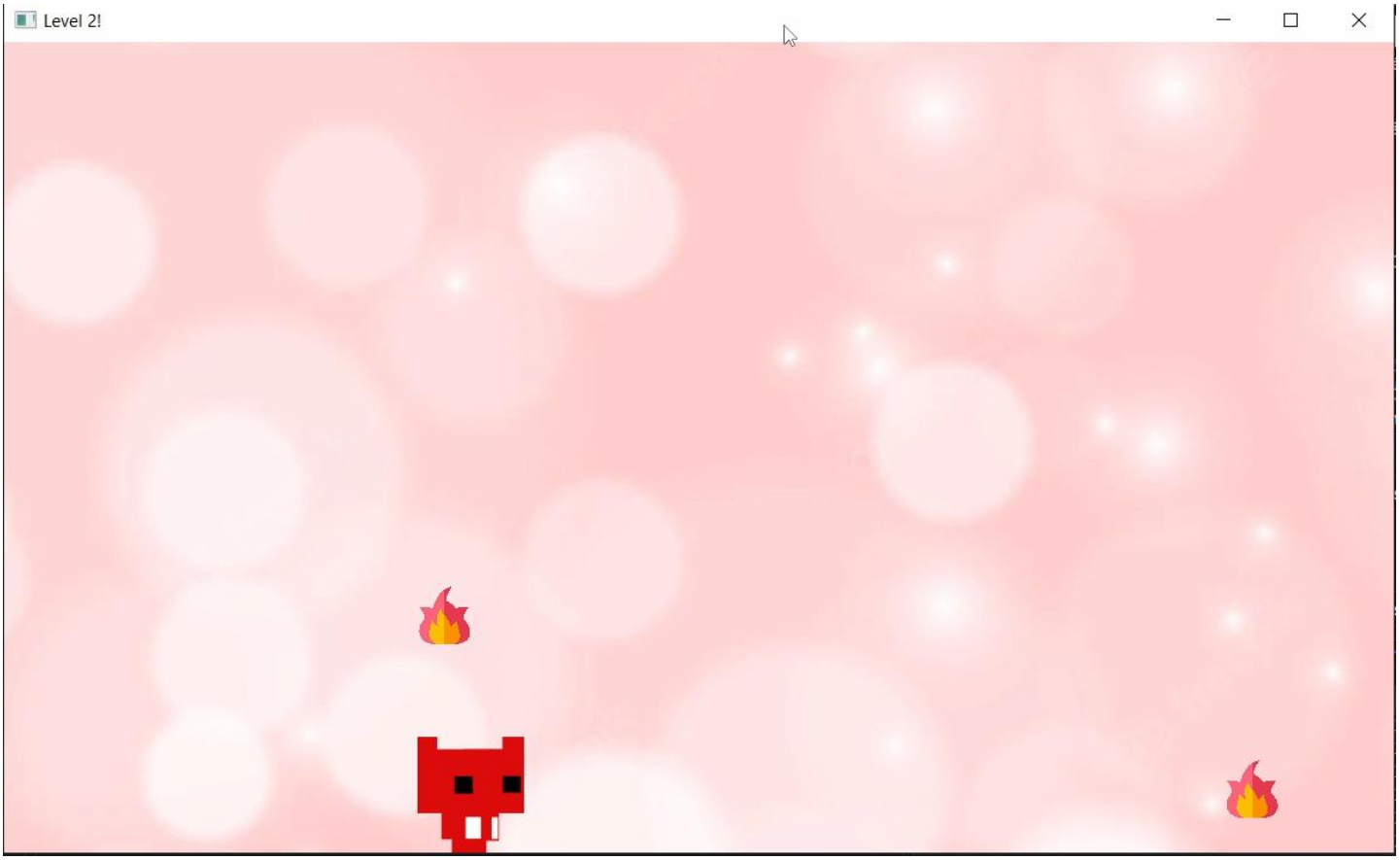
Alors vous affiche menu de jeux :

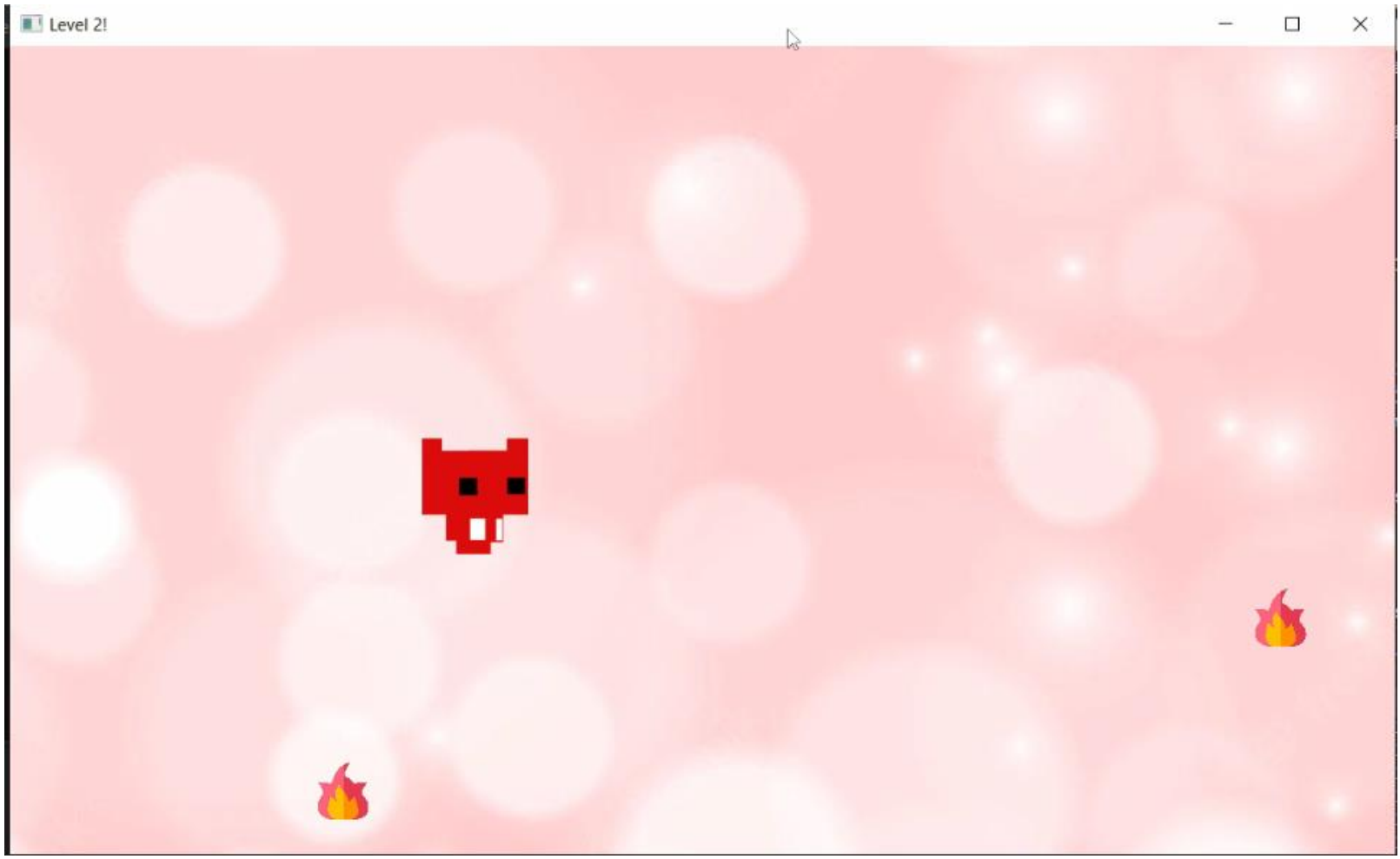


Et voilà maintenant vous devez choisir level 2 ou 1 o quelqu'un avec clavier et clic sur entrer

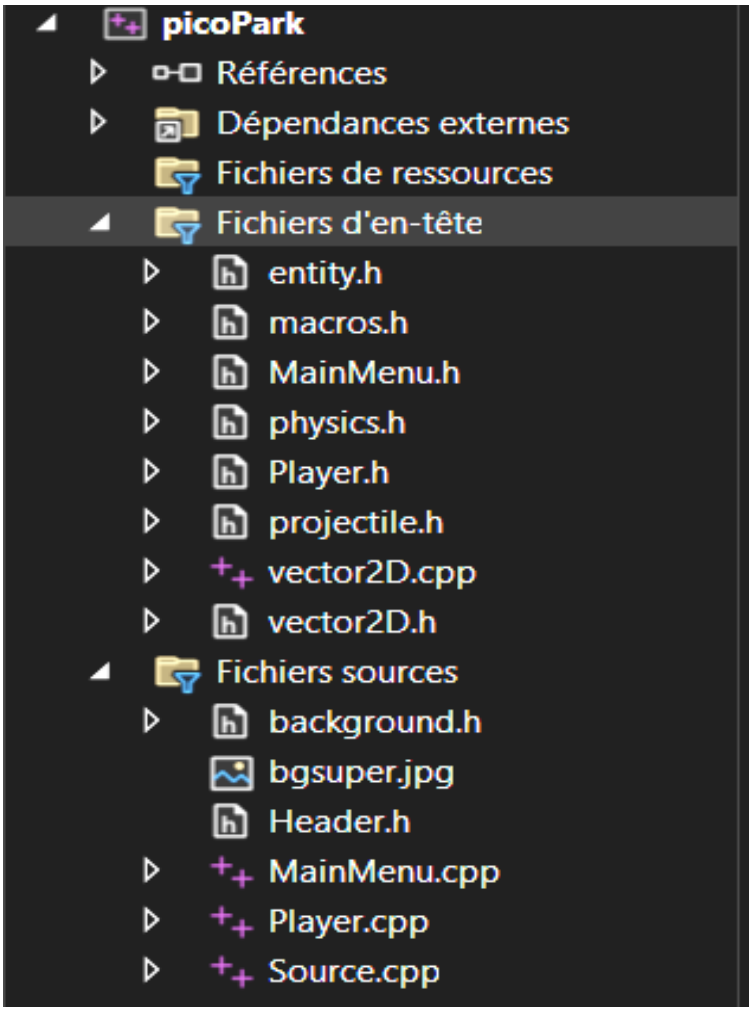
Example : clic sur Level 2




Level 2!


Level 2!

Code source :


picoPark
Références
Dépendances externes
Fichiers de ressources
Fichiers d'en-tête
entity.h
macros.h
MainMenu.h
physics.h
Player.h
projectile.h
vector2D.cpp
vector2D.h
Fichiers sources
background.h
bgsuper.jpg
Header.h
MainMenu.cpp
Player.cpp
Source.cpp

```cpp
#include <SFML/Graphics.hpp>

#include "Player.h"

namespace sf {
    class RenderWindow;
    class Event;
}

Player::Player() {

};

Player::~Player() {

};

void Player::inputs(const sf::Time& deltaTime) {
    if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
    {
        physics.location.x += speed * deltaTime.asSeconds();
    }
    if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
    {
        physics.location.x -= speed * deltaTime.asSeconds() ;
    }
    if (!isJumping) {
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
        {
            physics.velocity.y = - jumpSpeed;
            isJumping = true;
        }
    }
```

On a défini la classe *Player* avec les *constructeurs* et *déconstructeurs* puis on a fait la fonction qui permet aux joueurs de déplacer dans tous les directions droite, gauche, bas, haut utilisant la condition if pour les organiser.

On a affecté la vitesse de chaque pas dépend du temps en utilisant l'instruction qui se trouve dans sfml *asSeconds ()* aux positions de Player définie par location *vector2D x* et *y*. en utilisant l'opérateur + pour avancer en avant et l'opérateur - pour retourner à l'arrière.

On a attribué chaque saut une vitesse *de 1550.f* à l'instance Velocity définie dans *physics*.

Background.h :

```
agma once
clude <SFML/Graphics.hpp>

clude <vector>


ss Background

vate:

 std::vector<sf::Sprite> sprites;
  float speed = 0.1f;
lic:
 Background() = default;
 ~Background() = default;

 void addSprite(sf::Sprite newSprite)
 {
    if (sprites.size() != 0) {
        newSprite.setPosition(sprites[sprites.size() - 1].getPosition().x + sprites[sprites.size() - 1].getTexture()->getSize().x,

        }
            sprites.emplace_back(newSprite);


 }

 void draw(sf::RenderWindow& window)
 {

```

Pour la fond d'écran on a fait une classe nommé background qui la désigne par la fonction draw qui a réglé les mesures par le variable *spriteID* .

Il varie de 0 à la taille de l'image qu'on a ajouté par la fonction *addsprite*, après on vérifie que la largeur est positif et on affecte la position utilisant *getposition()* afin de désigner la fenêtre par *window.draw(sprite);*

Marcos.h

```
#pragma once
constexpr float WINDOW_SIZE_X = 1200;
constexpr float WINDOW_SIZE_Y = 700;


constexpr float gravityPower = 2300;

constexpr float minimalfiredelay = 0.7f;
```

- Dans macros on a précisé la taille de la fenêtre par
  WINDOW_SIZE_X = 1200
  WINDOW_SIZE_Y = 700
- *gravityPower* est responsable d'attirer le joueur en bas après le saut et pour *minimalfiredelay* c'est le minimal délai pour dépasser l'obstacle.

Source.cpp

Pour le 2^{er} niveau :

```
void addfire(std::vector<Projectile>& fireobs, std::vector<sf::Texture>& textures, float speed, bool isfireDown = true) {
    fireobs.emplace_back(Projectile());
    Projectile& fireob = fireobs[fireobs.size() - 1];
    fireob.speed = speed;
    fireob.setSprite(textures[1]);
    fireob.physics.location.x = WINDOW_SIZE_X;
    if (isfireDown) {
        fireob.physics.location.y = WINDOW_SIZE_Y - 80;
    }
    else {
        fireob.physics.location.y = WINDOW_SIZE_Y - 230;

    }

};
```

addfire est la fonction responsable de additionner les obstacles des feux . on précisé le positionnement soit horizontal ou verticale par *physics.location.x* , *physics.location.y* après l'on affecter une valeur de taille de fenêtre moins la valeur qui vous voulez . Cette fonction est pour le 2éme niveau.

Pour le 1^{er} niveau :

```
void adfire(std::vector<Projectile>& fireobs, std::vector<sf::Texture>& textures, float speed, bool isfireDown = true)
{
    fireobs.emplace_back(Projectile());
    Projectile& fireob = fireobs[fireobs.size() - 1];
    fireob.speed = speed;
    fireob.setSprite(textures[1]);
    fireob.physics.location.x = WINDOW_SIZE_X;

    fireob.physics.location.y = WINDOW_SIZE_Y - 100;


};
```

```
bool addBackgroundTexture(const std::string& filename, std::vector<sf::Texture>& textures)
{
    {
        textures.emplace_back(sf::Texture());
        sf::Image curImage;
        if (!curImage.loadFromFile(filename))
            return false;
        else {
            constexpr unsigned int  margin = 0.5;
            constexpr float transition = 3;
            for (unsigned int y = 0; y < curImage.getSize().y; y++)
            {

                for (unsigned int x = 0; x < margin; x++)
                {
                    sf::Color  color = curImage.getPixel(x, y);

                    unsigned int delta = float(margin - x) * transition;
                    if (delta > 255)
                        delta = 255;

                    if (color.r < 255 - delta)
                        color.r += delta;
                    else
                        color.r = 255;

                    if (color.g < 255 - delta)
                        color.g += delta;
                    else
                        color.g = 255;
```

La fonction *addBackgroundTexture* permet de faire une alternance de l'image de fond d'écran en précisé la valeur de margine qui le sépare.

```
//make a main window
RenderWindow MENU(VideoMode(960, 720), "Main Menu", Style::Default);
MainMenu mainMenu(MENU.getSize().x, MENU.getSize().y);
// set BACKGROUND
RectangleShape background;
background.setSize(Vector2f(960, 720));
Texture MainTexture;
MainTexture.loadFromFile("media/winter2.jpg");
background.setTexture(&MainTexture);
```

On a créé une fenêtre utilisant *RenderWindow* qui se trouve dans sfml et un fond d'écran pour le *menu* du jeux.

```
while (MENU.isOpen())
{
    Event event;
    while (MENU.pollEvent(event))
    {
        if (event.type == Event::Closed)
        {
            MENU.close();
        }
        if (event.type == Event::KeyReleased)
        {
            if (event.key.code == Keyboard::Up)
            {
                mainMenu.MoveUp();
                break;
            }

            if (event.key.code == Keyboard::Down)
            {
                mainMenu.MoveDown();
                break;
            }
        }
        if (event.key.code == Keyboard::Return)
        {
            RenderWindow Play(VideoMode(WINDOW_SIZE_X, WINDOW_SIZE_Y), "Level 2!");
            // RenderWindow Play(VideoMode(960, 720), "game_name");
            RenderWindow Options(VideoMode(WINDOW_SIZE_X, WINDOW_SIZE_Y), "Level 1");
            //RenderWindow About(VideoMode(WINDOW_SIZE_X, WINDOW_SIZE_Y), "About");

            int x = mainMenu.MainMenuPressed();
            if (x == 0)
```

On a appelé l'instance menu puis on a attribué *MENU.close();* losque l'utilisateur voulait fermé le menu .

mainMenu.MoveUp(); mainMenu.MoveDown(); : pour mouvez en bas et en haut .

```
Options.close();
// About.close();
Play.clear();
sf::Clock clock;
float totalTime = 0.f;

std::vector<sf::Texture> textures;
constexpr unsigned int nbTextures = 1;
textures.reserve(nbTextures);

textures.emplace_back(sf::Texture());
if (!textures[0].loadFromFile("media/player85-pp.png"))
    return EXIT_FAILURE;

textures.emplace_back(sf::Texture());
if (!textures[1].loadFromFile("media/feu-1.png"))
    return EXIT_FAILURE;

// sound
sf::Music music;
if (!music.openFromFile("media/padoru.wav"))
    return -1;
music.play();
music.setLoop(true);


addBackgroundTexture("media/rezde.png", textures);
addBackgroundTexture("media/rezde.png", textures);
```

sf::Clock clock; se trouve dans sfml responsable de définir le temps
sf::Music music; se trouve dans sfml pour le sens.
music.play(); music.setLoop(true); pour lire le sens qui vous avez entré.

On a fait un appel de fonction *addBackgroundTexture* pour ajouter les images de fond d'écran.

Conclusion

En fin on a réalisé le projet qui nous permet à voir la réalité et l'utilisation de la programmation orienté objet avec le langage c++ . D'après la réalisation de ce projet on a constaté que le développement se base sur la recherche et la pratiques et de tomber dans mer d'erreurs et les corrigées en fin on a beaucoup apprendre la patience, la recherche, esprit d'équipe.

Bibliothèques

- https://www.sfml-dev.org/
- https://www.sfmldev.org/download/sfml/2.5.1/
- https://www.sfml-dev.org/tutorials/2.5/
- https://www.sfml-dev.org/learn.php