Game.cpp

```
#include "pch.h"

// Global variables
namespace {
// Indexes
int i = 0;
bool showMenu = false;
// DVD Logo App
float f_xVelocity = 3.f;
float f_yVelocity = 3.f;
// Useful colors
const sf::Color LIGHT_GRAY = sf::Color(128, 128, 128);
const sf::Color DARK_GRAY = sf::Color(50, 50, 50);
const sf::Color DARKEST_GRAY = sf::Color(32, 32, 32);
}

// Constructor / Destructor
Game::Game() :
window(new sf::RenderWindow(sf::VideoMode(1200,715), "App SFML",
sf::Style::None)),
m_isRunning(true)
{
initFonts();
initTextures();
initTexts();
window->setVerticalSyncEnabled(true);
t0 = std::chrono::high_resolution_clock::now();
t1 = t0;
}
Game::~Game() {}

// Initialization
void Game::initFonts() {
Impact.loadFromFile("Assets/Fonts/impact.ttf");
Arial.loadFromFile("Assets/Fonts/arial.ttf");
Comic.loadFromFile("Assets/Fonts/comic.ttf");
}
void Game::initTextures() {
// Load from file
appIcon.loadFromFile("Assets/Images/icon.png");
logoDVD.loadFromFile("Assets/Images/Logos/Logo_DVD.png");
mus_megalovania.openFromFile("Assets/Songs/megalovania.ogg");
smallAppIcon.loadFromFile("Assets/Images/iconSmall.png");

// App's icon in the taskbar
window->setIcon(appIcon.getSize().x, appIcon.getSize().y,
appIcon.getPixelsPtr());
// App's icon in the top left corner
smallAppIconSpr.setTexture(smallAppIcon);
smallAppIconSpr.setPosition(5.f, 5.f);
```

```
// Colors init
sf::Color col1 (255, 0 , 0 ); sf::Color col2 (255, 128, 0 ); sf::Color col3
(255, 255, 0 );
sf::Color col4 (128, 255, 0 ); sf::Color col5 (0 , 255, 0 ); sf::Color col6 (0
, 255, 128);
sf::Color col7 (0 , 255, 255); sf::Color col8 (0 , 128, 255); sf::Color col9 (0
, 0 , 255);
sf::Color col10(128, 0 , 255); sf::Color col11(255, 0 , 255); sf::Color
col12(255, 0 , 128);
colors = {
col1, col2, col3, col4, col5, col6, col7, col8, col9, col10, col11, col12
};

// Shapes init
// Window boundaries (makes user experience better)
windowBounds.setPosition(1.f, 1.f);
windowBounds.setSize(sf::Vector2f((float)window->getSize().x - 2.f,
(float)window->getSize().y - 2.f));
windowBounds.setFillColor(sf::Color::Transparent);
windowBounds.setOutlineThickness(1.f);
windowBounds.setOutlineColor(sf::Color::Black);
// Topbar
CustomTitleBarBG.setOrigin(sf::Vector2f(0.f, 0.f));
CustomTitleBarBG.setSize(sf::Vector2f((float)window->getSize().x,
m_topBarHeight));
CustomTitleBarBG.setFillColor(sf::Color::White);
// Topbar buttons
windowExitCross.setPosition(window->getSize().x - 40.f, 0.f);
windowExitCross.setSize(sf::Vector2f(40.f, 30.f));
windowExitCross.setFillColor(sf::Color::White);
windowExitCrossRect.setPosition(window->getSize().x - 40.f, 0.f);
windowExitCrossRect.setSize(sf::Vector2f(38.f, 30.f));
windowExitCrossRect.setFillColor(sf::Color::Transparent);
windowExitCrossRect.setOutlineThickness(1.f);
windowExitCrossRect.setOutlineColor(sf::Color::Black);
// ---
windowReduceLine.setPosition(window->getSize().x - (2 * 40.f), 0.f);
windowReduceLine.setSize(sf::Vector2f(40.f, 30.f));
windowReduceLine.setFillColor(sf::Color::White);
windowReduceLineRect.setPosition(window->getSize().x - 79.f, 0.f);
windowReduceLineRect.setSize(sf::Vector2f(38.f, 30.f));
windowReduceLineRect.setFillColor(sf::Color::Transparent);
windowReduceLineRect.setOutlineThickness(1.f);
windowReduceLineRect.setOutlineColor(sf::Color::Black);
// Main application textures
sprLogoDVD.setTexture(logoDVD);
sprLogoDVD.scale(.4f, .4f);
sprLogoDVD.setPosition(sf::Vector2f( // Essentially placing it in the middle of
the window
((float)((window->getSize().x) / 2) - (float)
((sprLogoDVD.getGlobalBounds().width) / 2)),
((float)((window->getSize().y) / 2) - (float)
((sprLogoDVD.getGlobalBounds().height) / 2))
));
```

```
// Contextual menu when clicking the top left corner icon
menuBackground.setSize(sf::Vector2f(100, 60));
menuBackground.setFillColor(DARK_GRAY);
menuBackground.setOutlineThickness(1.f);
menuBackground.setOutlineColor(sf::Color::Black);
// Choices in that menu ^^^
menuTextOneRect.setSize(sf::Vector2f(100, 30));
menuTextTwoRect.setSize(sf::Vector2f(100, 30));
menuTextOneRect.setFillColor(DARK_GRAY);
menuTextTwoRect.setFillColor(DARK_GRAY);
}
void Game::initTexts() {
menuTextOne.setString("Fermer"); menuTextTwo.setString("Réduire");
menuTextOne.setCharacterSize(17); menuTextTwo.setCharacterSize(17);
menuTextOne.setFont(Impact); menuTextTwo.setFont(Impact);
menuTextOne.setFillColor(sf::Color::White);
menuTextTwo.setFillColor(sf::Color::White);

windowTitle.setFont(Arial);
windowTitle.setCharacterSize(12);
windowTitle.setFillColor(sf::Color::Black);
windowTitle.setPosition(30.f, 7.5f);
windowTitle.setString("App SFML");

FPString.setFont(Impact);
FPString.setCharacterSize(20);
FPString.setFillColor(sf::Color::White);
FPString.setPosition(5.f, (5.f + CustomTitleBarBG.getSize().y));

timeSinceStart.setFont(Impact);
timeSinceStart.setCharacterSize(20);
timeSinceStart.setFillColor(sf::Color::White);
timeSinceStart.setPosition(5.f, ((float)window->getSize().y -
CustomTitleBarBG.getSize().y));
}

// Converts float numbers to nanoseconds time duration
static std::chrono::nanoseconds durationToDuration(const float& time_s) {
return std::chrono::round(std::chrono::duration{ time_s });
}

// Game loop
void Game::run() {
std::chrono::steady_clock::time_point t1 =
std::chrono::high_resolution_clock::now();

const float timeStep = 1.f / 60.f; // 60 FPS
float timeAccumulator = 0.f;

while (m_isRunning) {
// Elapsed time since last frame
float f_elapsedTime = Clock.restart().asSeconds();

// --- RUNTIME SECONDS COUNTER --- //
t1 += durationToDuration(f_elapsedTime);
```

```
auto dt = 1.e-9 * std::chrono::duration_cast(t1 - t0).count();
int timeDuration = (int)dt;
timeSinceStart.setString("Time : " + std::to_string(timeDuration) + "s");

// --- FPS CALCULATION --- //
timeAccumulator += f_elapsedTime;
float fps = 1.f / f_elapsedTime;
// Set the FPS digit precision to 2
std::ostringstream oss;
oss << std::fixed << std::setprecision(2) << fps;
std::string FPSValue = oss.str();
FPString.setString("FPS : " + FPSValue);

// Events handling
pollEvents();

// Main game's loop
while (timeAccumulator >= timeStep) { // To get a consistent update time
update();
timeAccumulator -= timeStep;
}

// Rendering part
render();
}
}

// Get the events happening inside of the window
void Game::pollEvents() {
while (window->pollEvent(event)) {
switch (event.type) {
case sf::Event::Closed:
std::cout << "Fenetre fermee par l'utilisateur" << std::endl;
m_isRunning = false;
window->close();
break;
// Key pressed cases
case sf::Event::KeyPressed:
switch (event.key.scancode) {
case sf::Keyboard::Scan::Escape:
std::cout << "L'application a ete terminee par l'utilisateur a l'aide de la
touche " << sf::Keyboard::getDescription(event.key.scancode).toAnsiString() <<
"\n";
m_isRunning = false;
window->close();
break;
case sf::Keyboard::E: // E in AZERTY
mus_megalovania.play();
break;
case sf::Keyboard::R: // R in AZERTY
mus_megalovania.pause();
break;
default:
break;
}
```

```
// Mouse events
case sf::Event::MouseMoved:
if (m_isMouseDragging) {
if (m_lastDownX >= 0 && m_lastDownX <= window->getSize().x && m_lastDownY >= 0
&& m_lastDownY <= m_topBarHeight)
window->setPosition(window->getPosition() + sf::Vector2(event.mouseMove.x -
m_lastDownX, event.mouseMove.y - m_lastDownY));
}
if (menuTextOneRect.getGlobalBounds().contains((float)event.mouseMove.x,
(float)event.mouseMove.y))
menuTextOneRect.setFillColor(LIGHT_GRAY);
else
menuTextOneRect.setFillColor(DARK_GRAY);

if (menuTextTwoRect.getGlobalBounds().contains((float)event.mouseMove.x,
(float)event.mouseMove.y))
menuTextTwoRect.setFillColor(LIGHT_GRAY);
else
menuTextTwoRect.setFillColor(DARK_GRAY);
if (windowReduceLine.getGlobalBounds().contains((float)event.mouseMove.x,
(float)event.mouseMove.y))
windowReduceLine.setFillColor(sf::Color(46, 193, 255));
else
windowReduceLine.setFillColor(sf::Color::White);
if (windowExitCross.getGlobalBounds().contains((float)event.mouseMove.x,
(float)event.mouseMove.y))
windowExitCross.setFillColor(sf::Color(255, 67, 67));
else
windowExitCross.setFillColor(sf::Color::White);
break;
// Mouse clicks
case sf::Event::MouseButtonPressed:
if (event.mouseButton.button == sf::Mouse::Right) {
if (event.mouseButton.x >= 5 && event.mouseButton.x <= 25 &&
event.mouseButton.y >= 5 && event.mouseButton.y <= 25) {
showMenu = true;
menuBackground.setPosition(0.f, 30.f);
menuTextOneRect.setPosition(0.f, 30.f);
menuTextTwoRect.setPosition(0.f, 60.f);
menuTextOne.setPosition(10.f, 35.f);
menuTextTwo.setPosition(10.f, 65.f);
}
}
else if (event.mouseButton.button == sf::Mouse::Left) {
// "Fermer"
if (menuTextOneRect.getGlobalBounds().contains((float)event.mouseButton.x,
(float)event.mouseButton.y) ||
windowExitCross.getGlobalBounds().contains((float)event.mouseButton.x,
(float)event.mouseButton.y)) {
std::cout << "Fenetre fermee par l'utilisateur" << std::endl;
m_isRunning = false;
window->close();
}
// "Réduire"
else if (menuTextTwoRect.getGlobalBounds().contains((float)event.mouseButton.x,
(float)event.mouseButton.y) ||
windowReduceLine.getGlobalBounds().contains((float)event.mouseButton.x,
(float)event.mouseButton.y)) {
std::cout << "Fenetre reduite par l'utilisateur" << std::endl;
windowReduceLine.setFillColor(sf::Color::White);
HWND hwnd = window->getSystemHandle();
ShowWindow(hwnd, SW_MINIMIZE);
showMenu = false;
break;
}
else
showMenu = false; // Cacher le menu si on clique ailleurs
}
m_lastDownX = event.mouseButton.x;
m_lastDownY = event.mouseButton.y;
m_isMouseDragging = true;
break;
case sf::Event::MouseButtonReleased:
m_isMouseDragging = false;
break;
default:
break;
}
}
}

// Logic part of the game
void Game::update() {
// WINDOW
/* (0;0)
/--------------------------\
|--------------------------|
|FPS |
| |
| |DVD| |
|TIME |
\--------------------------/
(height; width) */
float xPos = sprLogoDVD.getPosition().x;
float yPos = sprLogoDVD.getPosition().y;
sprLogoDVD.move(f_xVelocity, f_yVelocity);
if (sprLogoDVD.getPosition().x < 0 || sprLogoDVD.getPosition().x >(window-
>getSize().x - sprLogoDVD.getGlobalBounds().width)) {
f_xVelocity *= -1;
++i;
}
if (sprLogoDVD.getPosition().y < m_topBarHeight || sprLogoDVD.getPosition().y >
(window->getSize().y - sprLogoDVD.getGlobalBounds().height)) {
f_yVelocity *= -1;
++i;
}
if (i % 12 == 0) i = 0;
sprLogoDVD.setColor(colors[i]);
}
```

```
// Rendering function
void Game::render() {
// Clear the window to a dark-themed background (#1E1E1E in hex)
window->clear(DARKEST_GRAY);

// Drawing part
window->draw(sprLogoDVD);
// end
window->draw(FPString);
window->draw(timeSinceStart);

if (showMenu) {
window->draw(menuBackground);
window->draw(menuTextOneRect);
window->draw(menuTextOne);
window->draw(menuTextTwoRect);
window->draw(menuTextTwo);
}

// Last layer elements separated from the other elements to avoid layering
problems
window->draw(CustomTitleBarBG);
window->draw(windowTitle);
window->draw(windowReduceLine);
window->draw(windowExitCross);
window->draw(windowReduceLineRect);
window->draw(windowExitCrossRect);
window->draw(smallAppIconSpr);
window->draw(windowBounds);

window->display();
}
```