Міністерство освіти і науки України

Національний технічний університет

«Харківський політехнічний інститут»

Факультет комп’ютерних наук і програмної інженерії

Кафедра інформатики та інтелектуальної власності (ІІВ)

**КУРСОВА РОБОТА**

з дисципліни «ООП»

Тема

"SFML гра"

Виконав

студент групи КН– 320Б Олексій ДРАБЧАК

Перевірив

ст. викладач кафедри ІІВ Максим СОБОЛЬ

Харків 2021

**РЕФЕРАТ**

Пояснювальна записка до програми: 50с., 9 рис., 5 джерел., 6 додатків.

Ключові слова:ООП, MICROSOFT VISUAL STUDIO 2019, SFML, ФРЕЙМВОРК.

Предмет дослідження: сучасні методи розробки діалогового вікна, а також застосування основних патернів ООП.

Завдання: опанувати набуті навички та знання з курсу ООП, опанувати фреймворк SFML та базові роботи з графікою і звуками, створивши просту гру в Visual Studio 2019.

Мета: курсової роботи: систематизація, поглиблення і активне застосування знань з програмування, закріплення знань, отриманих в лекційному курсі, а також отримання навичок в роботі з SFML фреймворк

ЗМІСТ

[ВСТУП 4](#_Toc90659524)

[ПЕРЕЛІК ПОЗНАК ТА СКОРОЧЕНЬ 5](#_Toc90659525)

[1 Аналіз предметної області 6](#_Toc90659526)

[1.1 Середовище програмування visual studio 6](#_Toc90659527)

[1.2 Графічна бібліотека SFML 7](#_Toc90659528)

[1.3 Інструкція як користуватися програмою 7](#_Toc90659529)

[2 Розробка програми 10](#_Toc90659530)

[2.1 Створення діалогового вікна 10](#_Toc90659531)

[2.2 Створення елементів діалогового вікна 11](#_Toc90659532)

[2.3 Опис використовуваних методів і подій 11](#_Toc90659533)

[2.4 Структура програми і основні функції 11](#_Toc90659534)

[3 Опис програмної реалізації 15](#_Toc90659535)

[3.1 Інтерфейс користувача 15](#_Toc90659536)

[3.2 Інструкція користувача 16](#_Toc90659537)

[ВИСНОВКИ 19](#_Toc90659538)

[Список джерел інформації 20](#_Toc90659539)

[Додаток А 21](#_Toc90659540)

[Додаток Б 25](#_Toc90659541)

[Додаток В 26](#_Toc90659542)

[Додаток Г 42](#_Toc90659543)

[Додаток Д: 48](#_Toc90659544)

[Додаток Є: 49](#_Toc90659545)

# ВСТУП

Основне завдання в роботі це створити додаток в діалоговому вікні, при цьому в коді використовувати ООП принципи.

Об'єктно– орієнтоване програмування парадигма програмування, в якій основними концепціями є поняття об'єктів і класів.

У центрі ООП знаходиться поняття об'єкта. Об'єкт– це сутність, екземпляр класу, якій можна посилати повідомлення і яка може на них реагувати, використовуючи свої дані. Дані об'єкта приховані від решти програми. Приховування даних називається інкапсуляцією. Наявність інкапсуляції досить для об'єктно мови програмування, але ще не означає його об'єктної орієнтованості– для цього потрібна наявність успадкування. Але навіть наявність інкапсуляції і спадкування не робить мову програмування в повній мірі об'єктним з точки зору ООП. Основні переваги ООП проявляються тільки в тому випадку, коли в мові програмування реалізований поліморфізм, тобто можливість об'єктів з однаковою специфікацією мати різну реалізацію. Так само в ООП часто рекомендують виділяти ще одну важливу характеристику – абстракцію. Офіційно її не вносили в обов'язкові риси ООП, але списувати її з рахунків не варто.

Абстрагування – це спосіб виділити набір значущих характеристик об'єкта, виключаючи з розгляду не значимі Відповідно, абстракція – це набір всіх таких характеристик.

Інкапсуляція – це властивість системи, що дозволяє об'єднати дані і методи, що працюють з ними в класі, і приховати деталі реалізації від користувача.

Спадкування **–**  це властивість системи, що дозволяє описати новий клас на основі вже існуючого з частково або повністю запозичує функціональністю. Клас, від якого виробляється спадкування, називається базовим, батьківським або суперкласом. Новий клас – нащадком, спадкоємцем або похідним класом

Поліморфізм – це властивість системи використовувати об'єкти з однаковим інтерфейсом без інформації про тип і внутрішню структуру об'єкта.

# ПЕРЕЛІК ПОЗНАК ТА СКОРОЧЕНЬ

UI – user interface (інтерфейс гравця).

# 1 Аналіз предметної області

## 1.1 Середовище програмування visual studio

Microsoft Visual Studio ­ це інтегроване середовище розробки (IDE) від Microsoft. Він використовується для розробки комп'ютерних програм, а також веб– сайтів, веб– програм, веб– сервісів та мобільних додатків.

Visual Studio дає змогу розробнику написати єдину програму для роботи на кількох платформах Windows, таких як мобільний, настільний і навіть експериментальний середовище Microsoft HoloLens. Він також забезпечує спосіб створення додатків, які взагалі не працюють на комп'ютерах Windows, але замість цього вони працюють на пристроях iOS або у веб– додатках у хмарі. 30 Visual Studio включає в себе редактор коду, що підтримує IntelliSense (компонент завершення коду), а також рефакторинг коду. Вбудований відладчик працює як відладчик на рівні вихідного рівня, так і відладчик на рівні машини. Інші вбудовані інструменти включають програму для кодування, конструктор форм для побудови графічних інтерфейсів, веб– дизайнер, дизайнер класів та дизайнер схеми баз даних. Він приймає плагіни, які покращують функціональність практично на всіх рівнях, включаючи підтримку систем керування джерельними ресурсами (наприклад, Subversion та Gіt) та додавання нових наборів інструментів, таких як редактори та візуальні розробники для мов або наборів інструментів для інших аспектів розробки програмного забезпечення життєвий цикл (наприклад, клієнт Team Foundation Server: Team Explorer). Visual Studio включає в себе редактор коду, що підтримує IntelliSense (компонент завершення коду), а також рефакторинг коду. Вбудований відладчик працює як відладчик на рівні вихідного рівня, так і відладчик на рівні машини. Інші вбудовані інструменти включають програму для кодування, конструктор форм для побудови графічних інтерфейсів, веб– дизайнер, дизайнер класів та дизайнер схеми баз даних. Він приймає плагіни, які покращують функціональність практично на всіх рівнях, включаючи підтримку систем керування джерельними ресурсами (наприклад, Subversion та Gіt) та додавання нових наборів інструментів, таких як редактори та візуальні розробники для мов або наборів інструментів для інших аспектів розробки програмного забезпечення життєвий цикл (наприклад, клієнт Team Foundation Server: Team Explorer). Visual Studio підтримує 36 різних мов програмування і дозволяє редакторові коду та відладчику підтримувати (в тій чи іншій мірі) майже будь– яку мову програмування, якщо існує певна мова– служба. Вбудовані мови включають C, C ++, C ++ / CLI, Visual Basic. NET, C #, F #, JavaScript, TypeScript, XML, XSLT, HTML та CSS.

## 1.2 Графічна бібліотека SFML

Бібліотека SFML надає простий інтерфейс для різних компонентів вашого комп'ютера, щоб полегшити розробку ігор і мультимедійних застосувань. Вона складається з п'яти модулів:

1. system – управління часом і потоками, він є обов'язковим, оскільки усі модулі залежать від нього;
2. window – управління вікнами і взаємодією з користувачем;
3. graphics – робить простим відображення графічних примітивів і зображень;
4. audio – надає інтерфейс для управління звуком;
5. network – для мережевих застосувань.

Використовуючи SFML, програма може бути скомпільоване і запущена на найбільш поширених платформах: Windows, Linux, Mac. SFML офіційно підтримує C і .NET. Також, завдяки своєму активному співтовариству, вона доступна на багатьох інших мовах, таких як Java, Ruby, Python, Go та ін.

## 1.3 Інструкція як користуватися програмою

Програма має стартове меню в якому в свою чергу присутні дві кнопки: Play – запускає гру, Exit – виходить з гри.

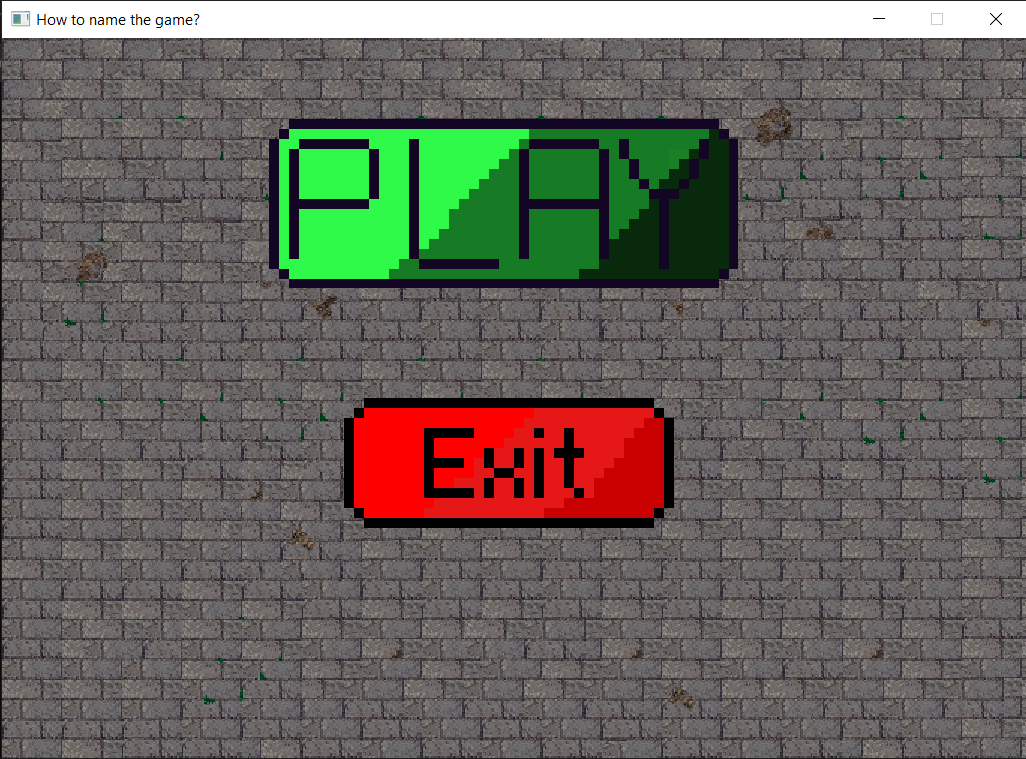


Рисунок 1.1 – Ігрове меню



Рисунок 1.2 – UI гравця

Керування персонажем здійснюється в два способи:

1. Клавіші W,A,S,D (W – вперед, A – вліво, S – вправо, D – вниз).
2. Клавіші ↑, ↓, →, ← (↑ – вперед, ↓ – вниз, → – вправо, ← – вліво).

Головне завдання гри – зібрати якомога більше монеток. При зіткненні з ворогом здоров’я зменшується, при зіткненні з ресурсом здоров’я відновлюється на певну величину.

Гра закінчується коли в гравця закінчується здоров’я.

Після закінчення здоров’я з’являється відповідний напис про його закінчення та кнопка Restart, натиснувши яку гра перезапускається.



Рисунок 2.3 – закінчення гри

Щоби вийти з гри раніше ніж закінчиться здоров’я натисніть Esc клавішу або просто закрийте вікно.

# 2 Розробка програми

## 2.1 Створення діалогового вікна

Для малювання об'єктів, що надаються графічним модулем, необхідно використовувати спеціалізований клас вікна: RenderWindow. Цей клас є похідним від Window і успадковує всі його функції. Все, що ви дізналися про sf :: Window (створення, обробка подій, управління частотою кадрів, змішування з OpenGL і т.д.), можна застосувати і до RenderWindow.

Крім того, RenderWindow додає високо рівневі функції, щоб допомогти вам легко малювати. У цьому підручнику ми зосередимося на двох з цих функцій: clear і draw. Вони прості, як і випливає з їх назви: clear очищає все вікно вибраним кольором, а draw малює будь– який об'єкт, який ви йому передаєте.

Усі методи класу RenderWindow реалізовані в методі render(); класу MenuGUI програми поданий у додатку А. Виклик методу render() є обов'язковим, інакше об’єкти не будуть відображатися в вікні. Виклик clear перед малюванням чого– небудь є обов'язковим, інакше вміст попередніх кадрів буде присутній за всім, що ви намалюєте. Єдиний виняток – коли ви закриваєте всі вікно тим, що малюєте, так що жоден піксель не буде відмалювали. В цьому випадку ви можете не викликати clear (хоча це не зробить помітного впливу на продуктивність).

Виклик display також обов'язковий, він бере те, що було намальовано з моменту останнього дзвінка display, і відображає це на вікні. Насправді, все малюється не прямо в вікно, а в прихований буфер. Цей буфер потім копіюється в вікно при виклику display – це називається подвійний буферизацією.

## 2.2 Створення елементів діалогового вікна

У SFML можливо зробити різні фігури і не тільки, для цього потребується створити об’єкт класу потрібної фігури. Щоб сформувати прямокутну кнопку у вікні, створюється об’єкт класу RectangleShape, а у конструктор з параметрами передається клас Vector2f(float X, float Y), де х ­­­­­­­­­– висота, у – ширина фігури. Далі використовуючи методи класу, задається колір фігурі setFillColor(), якому у параметри передається клас Color(), який приймає у параметри три цифри RGB та четвертий параметр Alfa так звану прозорість об’єкта. Після цього задається позиція фігури функцією setPosition(float x, float y), де х ¬ ширина, у – висота. Після налаштування, фігура підмальовується на вікні, для цього викликається метод draw() об’єкту window, у параметри функції передається назва фігури.

## 2.3 Опис використовуваних методів і подій

У бібліотеці SFML є клас, який обробляє події, а саме натиснення клавішею мишки або клавіатури, кнопки джойстика и тд. Клас називається Event. За допомогою об'єкту цього класу, робиться взаємодія користувача з вікном і його елементами. Наприклад, при натисненні кнопки мишки, об'єкту класу Event вже зберігає про це інформацію. Таким чином можливо побудувати умову if () в якому буде порівняння об'єкту класу Event і потрібної дії, якщо вони рівні програма заходить в умову. Реалізацію методу обробки подій наведено в додатку А.

## 2.4 Структура програми і основні функції

Початок програми починається з функції main, де ми маємо об’єкт класу Menu який в своючергу виклакає цикл з головним меню дивись в додатку В.

Клас Game відповідає за всі події, основні механіки гри, їх обробку та відображення всіх подій у вікні. Даний клас був створений таким чином, щоб розділити процеси, як от: оновлення, зміна та перевірка даних при кожній ітерації, в коді цього класу цей метод має назву Update(), та відображення всіх об’єктів у методі Render(), які беруть візуальну і не тільки участь у програмі. В свою чергу дані методи об’єднують в собі методи update(), render() інших класів, об’єкти яких потребують зміни своїх властивостей, параметрів або відображення у вікні відповідно. Таким чином підвищується читабельність коду та в загалом логіки програми. Даний метод був взятий з книги «Game Programming Patterns» by Robert Nystrom.

Клас Game застосовує принцип композиції об’єктів та навіть агрегації: містить в собі об’єкти та вказівники на об’єкти та вектори вказівників на об’єкти інших класів.

Також, даний клас містить приватні методи для ініціалізації певних об’єктів, змінних, текстур, елементів ігрового інтерфейсу, векторів з вказівниками на об’єкти і тощо. Такий підхід дає змогу написати наступний конструктор який наведено в додатку С.

Не менш важливими є класи для опису об’єкта гравця Player та клас для опису ворога SwagBall а також інтерфейси для опису поводження об’єктів класів гравця та ворога. Класи Player та SwagBall в свою чергу успадковуються від інтерфейсів InterfacePlayer та InterfaceSwagBall.



Рисунок 2.1



Рисунок 2.2



Рисунок 2.3



Рисунки 2.1 – 2.4 – процес гри

# 3 Опис програмної реалізації

## 3.1 Інтерфейс користувача

Гравець має шкалу здоров’я яке при взаємодії з об’єктом ворог – зменшується а при взаємодії з об’єктом здоров’я – відновлюється. А також є шкала балів яка збільшуються при взаємодії з об’єктом бал.

Керування гравця здійснюється за допомогою клавіш (W, A, S, D) та

(↓, ←, →, ↑). Пересування гравця супроводжується анімацією руху.

Гру можна завершити двома способами:

1. Коли закінчиться здоров’я;
2. Натиснути клавішу Esc;
3. Закрити вікно з грою.

При закінченні здоров’я користувачу виводиться відповідне повідомлення та завершується гра, а також є змога заново почати гру натиснувши кнопку Restart.

## 3.2 Інструкція користувача

Щоб усе працювало, потрібно підключити бібліотеку SFML. Для цього завантажуються бібліотеки sfml– graphics 2.5.1 та sfml– audio 2.5.1. Відкрийте Visual studio та створюється пустий проект, далі в пошуку пишемо NuGet.

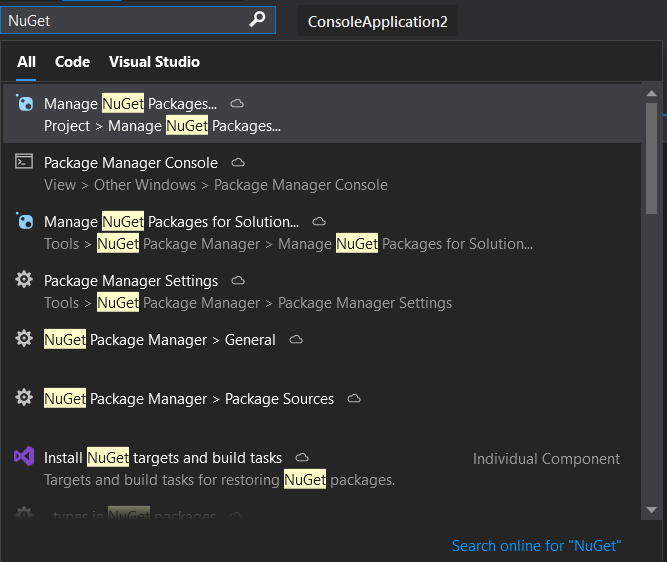


Рисунок 3.1 – встановлення бібліотек SFML

Далі пишемо в пошуку відкритого вікна спочатку sfml– graphics 2.5.1 потім sfml– audio 2.5.1. І встановлюємо по черзі.

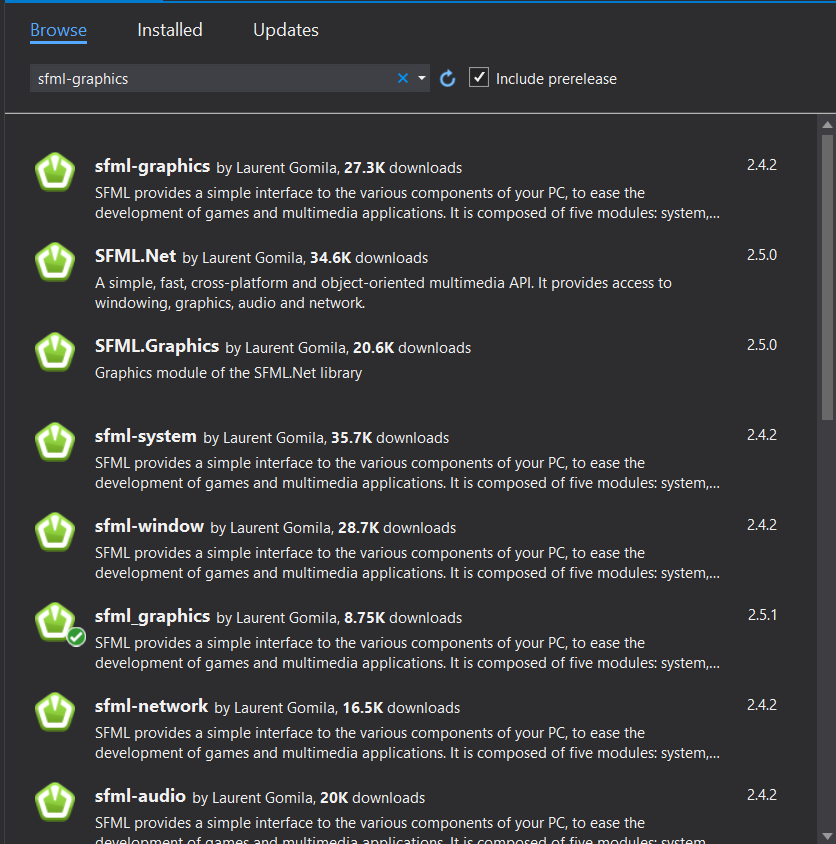


Рисунок 3.2 – встановлення бібліотек

Слідуйте інструкції:

Правою кнопкою миші по проекту → Properties→General→Platform Toolset→

→Visual Studio 2017(v141).

Далі Configuration Properties→ General→ Window SDK Version→ 10.0.19041.0→ Apply→Ok

І можна користуватися SFML Framework.

У самому проекті підключаються бібліотеки #include<SFML/Graphics.hpp>

#include<SFML/System.hpp>

#include<SFML/Window.hpp>

#include<SFML/Audio.hpp>

#include<SFML/Network.hpp>, та його простір using namespace sf;

# ВИСНОВКИ

У цій роботі були зроблена гра. В ході розробки гри була освоєна бібліотека SFML і закріплені базові знання щодо її використання. Так само закріплені навички програмування, які були отримані під час навчання і написання лабораторних робіт.

У курсовій роботі були використані класи, а так само контейнери. Ще в проекті є можливість спостерігати ООП принципи на практиці.

У грі є можливість додавання нових рівнів з найменшим зміною коду. Так само були помічені недоробки, в плані, тільки один тип монстрів, монстри не вміють ухилятися від атак гравця.

За допомогою практичної частини курсового проектування були закріплені теоретичні знання, що ми освоїли за курс ООП.

# Список джерел інформації

1. Дмитрий Бушуев: Вступ і установка графічної бібліотеки SFML в C++//[https://ravesli.com/graficheskaya– biblioteka– sfml– vstuplenie– i– ustanovka/](https://ravesli.com/graficheskaya-biblioteka-sfml-vstuplenie-i-ustanovka/), 23.04.2021
2. Laurent Gomila: SFML. Основний сайт графічної бібліотеки // [https://www.sfml– dev.org](https://www.sfml-dev.org), 05.05.2021
3. «Game Programming Patterns» by Robert Nystrom ­­­ – 2011. – 321 c.
4. Suraj Sarma SFML Simple Games Guide

<https://www.youtube.com/c/SurajSharmaFourKnob/playlists>.

1. Посилання на вихідний код програми в GitHub

https://github.com/whoiam– bat/CourseProject– 2course

# Додаток А

**Повний код реалізації класу меню.**

Menu.h:

#pragma once

#include "libraries.h"

#include "Game.h"

#include "SFML/Audio/Sound.hpp"

#include "SFML/Audio.hpp"

class MenuGUI {

private:

RenderWindow\* window;

VideoMode videoMode;

Vector2i mousePosition;

Vector2f mousePosView;

bool isMenuRuns;

bool isClickOnPlay;

bool isClickOnExit;

//Animation

Texture backgroundTexture;

Texture playButtonTexture;

Texture exitButtonTexture;

Sprite background;

Sprite playButton;

Sprite exitButton;

//Sounds and music

SoundBuffer clickBuffer;

Sound clickSound;

Music backgroundSound;

//Private functions

void initWindow();

void initTextures();

void initSprites();

void initVariables();

void initSounds();

public:

//Constructors and destructors

MenuGUI();

~MenuGUI();

//Accessors

const bool& running() const;

//Functions

void runMenu();

void pollEvents();

void updateUserCollisionWithButtons();

void updateMousePosition();

void update();

void renderGUI();

void render();

};

Menu.cpp

#include "MenuGUI.h"

void MenuGUI::initWindow() {

this– >videoMode = VideoMode(1024, 720);

this– >window = new RenderWindow(this– >videoMode, "How to name the game?", Style::Close | Style::Titlebar);

//Defines how fast will move objects in the window

this– >window– >setFramerateLimit(144);

this– >window– >setVerticalSyncEnabled(false);

}

void MenuGUI::initTextures() {

this– >backgroundTexture.loadFromFile("images/floor.png");

this– >playButtonTexture.loadFromFile("images/play.png");

this– >exitButtonTexture.loadFromFile("images/exit.png");

}

void MenuGUI::initSprites() {

//Set starting parameters background sprite

this– >background.setTexture(this– >backgroundTexture);

this– >background.setPosition(Vector2f(– 60, – 60));

this– >background.setScale(Vector2f(1.5, 1.5));

//Set starting parameters for UI bttons

this– >playButton.setTexture(this– >playButtonTexture);

this– >playButton.setPosition(Vector2f(

this– >window– >getSize().x / 2.f – this– >playButton.getGlobalBounds().width / 2.4f ,

this– >window– >getSize().y / 2.f – this– >playButton.getGlobalBounds().height

));

this– >playButton.setTextureRect(IntRect(59, 49, 470, 170));

this– >exitButton.setTexture(this– >exitButtonTexture);

this– >exitButton.setPosition(Vector2f(

this– >window– >getSize().x / 2.f – this– >exitButton.getGlobalBounds().width / 2.f + 5,

this– >window– >getSize().y / 2.f – this– >exitButton.getGlobalBounds().top

));

this– >exitButton.setTextureRect(IntRect(10,10,330,130));

}

void MenuGUI::initVariables() {

this– >isClickOnExit = false;

this– >isClickOnPlay = false;

this– >isMenuRuns = true;

}

void MenuGUI::initSounds() {

if (!this– >clickBuffer.loadFromFile("melodies/button\_click.ogg")) {

cout << " ! ERROR::GAME::INITSOUNDS::COULD NOT LOAD SOUND\n";

}

if (!this– >backgroundSound.openFromFile("melodies/menu.wav")) {

cout << " ! ERROR::GAME::INITSOUNDS::COULD NOT LOAD MUSIC\n";

}

this– >backgroundSound.setVolume(30.f);

this– >clickSound.setBuffer(this– >clickBuffer);

this– >clickSound.setVolume(20.f);

}

//Constructors and destructors

MenuGUI::MenuGUI() {

this– >initWindow();

this– >initTextures();

this– >initSprites();

this– >initVariables();

this– >initSounds();

}

MenuGUI::~MenuGUI() {

delete this– >window;

}

//Accessors

const bool& MenuGUI::running() const {

return this– >window– >isOpen();

}

//Functions

void MenuGUI::runMenu() {

this– >backgroundSound.play();

while (this– >running()) {

if (this– >isClickOnPlay == true) {

this– >clickSound.play();

this– >backgroundSound.pause();

Game game;

this– >window– >close();

game.runGame();

}

if (this– >isClickOnExit == true) {

this– >window– >close();

}

else {

this– >update();

this– >render();

}

}

}

void MenuGUI::pollEvents() {

Event ev;

while (this– >window– >pollEvent(ev)) {

switch (ev.type) {

case (Event::Closed): {

this– >window– >close();

break;

}

case (Event::KeyPressed): {

if (ev.key.code == Keyboard::Escape) {

this– >window– >close();

}

break;

}

}

}

}

void MenuGUI::updateMousePosition() {

/\*\*\*

\* Updates the mouse positions:

\* relative to the window

\*/

this– >mousePosition = Mouse::getPosition(\*this– >window);

//Takes mouse position on screen and converts pixels to coordinates and writes in mousePosView variable

this– >mousePosView = this– >window– >mapPixelToCoords(this– >mousePosition);

}

void MenuGUI::updateUserCollisionWithButtons() {

if (Mouse::isButtonPressed(Mouse::Left)) {

if (this– >playButton.getGlobalBounds().contains(this– >mousePosView)) {

this– >isClickOnPlay = true;

}

else if (this– >exitButton.getGlobalBounds().contains(this– >mousePosView)) {

this– >isClickOnExit = true;

}

}

}

void MenuGUI::update() {

this– >pollEvents();

this– >updateMousePosition();

this– >updateUserCollisionWithButtons();

}

void MenuGUI::renderGUI() {

this– >window– >draw(this– >playButton);

this– >window– >draw(this– >exitButton);

}

void MenuGUI::render() {

this– >window– >clear();

//Render background

this– >window– >draw(this– >background);

//Render UI buttons

this– >renderGUI();

this– >window– >display();

}

# Додаток Б

**Повний код реалізації main функції.**

#include "Game.h"

#include "MenuGUI.h"

int main() {

//Initialize random seed

srand(static\_cast<unsigned>(time(NULL)));

//Initialize game object

MenuGUI menu;

//Game loop

menu.runMenu();

return 0;

}

# Додаток В

**Повний код реалізації ігрового класу.**

Game.h

#pragma once

#include "Player.h"

#include "SwagBall.h"

class Game {

private:

VideoMode videoMode;

RenderWindow\* window;

Player player;

string path;

Vector2i mousePositionPixels;

Vector2f mousePositionCoordinates;

//Player GUI

RectangleShape healthBar;

RectangleShape healthBarBack;

Font fontGUI;

Font gameOver;

Text guiText;

Text healthText;

Text endGameText;

Texture backgroundTexture;

Sprite background;

Texture restartTexture;

Sprite restartButton;

vector<SwagBall\*> swagBalls;

//Variables for spawning balls

float spawnTimerMax;

float spawnTimer;

int maxSwagBalls;

int points;

int rateScore;

//For control amount of damaging and healing enemies

int amountDamaging;

int maxAmountDamaging;

int amountHealling;

int maxAmountHealling;

//For restart button

bool isClickOnRestart;

//For ending game

bool endGame;

bool isBeatRecord;

//Sounds

SoundBuffer restartBuffer;

Sound restartSound;

SoundBuffer healBuffer;

Sound healSound;

SoundBuffer damagBuffer;

Sound damagSound;

SoundBuffer coinBuffer;

Sound coinSound;

SoundBuffer deathBuffer;

Sound deathSound;

SoundBuffer recordBuffer;

Sound recordSound;

//Private functions

void initSprite();

void initTexture();

void initFonts();

void initGUI();

void initVariables();

void initWindow();

void initSounds();

public:

//Constructors and Destructors

Game();

~Game();

void runGame();

//Accessors

const bool& getEndGame() const;

//Functions

void setScore();

const bool running() const;

void spawnSwagBalls();

const int randBallType();

void pollEvents();

void updateMousePosition();

void updateUserCollisionWithButtons();

void updateGUI();

void updateCollision();

void updatePlayer();

void update();

void renderGUIButtons();

void renderGUI(RenderTarget\* tagret);

void render();

};

Game.cpp

#include "Game.h"

void Game::initTexture() {

this– >backgroundTexture.loadFromFile("images/floor.png");

this– >restartTexture.loadFromFile("images/restart.png");

}

void Game::initSprite() {

//Set background sprite

this– >background.setTexture(this– >backgroundTexture);

this– >background.setPosition(Vector2f(– 60, – 60));

this– >background.setScale(Vector2f(1.5, 1.5));

//Set restart button

this– >restartButton.setTexture(this– >restartTexture);

this– >restartButton.setTextureRect(IntRect(20, 20, 470, 130));

this– >restartButton.setScale(Vector2f(0.4, 0.4));

this– >restartButton.setPosition(Vector2f(

this– >window– >getSize().x / 2.f – this– >restartButton.getGlobalBounds().width / 2.f,

this– >window– >getSize().y / 2.f

));

}

void Game::initFonts() {

if (!this– >fontGUI.loadFromFile("Fonts/gui.ttf")) {

cout << " ! ERROR::GAME::INITFONTS::COULD NOT LOAD gui.ttf\n";

}

if (!this– >gameOver.loadFromFile("Fonts/game– over.ttf")) {

cout << " ! ERROR::GAME::INITFONTS::COULD NOT LOAD game– over.ttf\n";

}

}

void Game::initGUI() {

//GUI text init

this– >guiText.setFont(this– >fontGUI);

this– >guiText.setPosition(Vector2f(static\_cast<float>(this– >window– >getSize().x – 330.f), 6.f));

this– >guiText.setFillColor(Color::White);

this– >guiText.setCharacterSize(26);

//Health text init

this– >healthText.setFont(this– >fontGUI);

this– >healthText.setPosition(Vector2f(static\_cast<float>(this– >healthBarBack.getGlobalBounds().width / 2.f + 100.f), 4.f));

this– >healthText.setFillColor(Color::White);

this– >healthText.setCharacterSize(20);

// Init player GUI health points

this– >healthBar.setSize(Vector2f(300.f, 25.f));

this– >healthBar.setFillColor(Color::Red);

this– >healthBar.setPosition(Vector2f(5.f, 5.f));

this– >healthBarBack = this– >healthBar;

this– >healthBarBack.setFillColor(Color(25, 25, 25, 200));

//Game over text init

this– >endGameText.setFont(this– >gameOver);

this– >endGameText.setStyle(Text::Bold);

this– >endGameText.setFillColor(Color::Red);

this– >endGameText.setCharacterSize(100);

this– >endGameText.setString("Game Over!");

this– >endGameText.setPosition(

this– >window– >getSize().x / 2.f – this– >endGameText.getGlobalBounds().width / 2.f,

this– >window– >getSize().y / 2.f – this– >endGameText.getGlobalBounds().height \* 2

);

}

void Game::initVariables() {

//Set player starting position

this– >isClickOnRestart = false;

this– >endGame = false;

this– >spawnTimerMax = 10.f;

this– >spawnTimer = this– >spawnTimerMax;

this– >maxSwagBalls = 10;

this– >points = 0;

this– >path = "Score.txt";

this– >setScore();

this– >isBeatRecord = true;

this– >maxAmountDamaging = 4;

this– >maxAmountHealling = 3;

this– >amountDamaging = 0;

this– >amountHealling = 0;

}

void Game::initWindow() {

this– >videoMode = VideoMode(1024, 720);

this– >window = new RenderWindow(this– >videoMode, "How to name the game?", Style::Close | Style::Titlebar);

//Defines how fast will move objects in the window

this– >window– >setFramerateLimit(144);

this– >window– >setVerticalSyncEnabled(false);

}

void Game::initSounds() {

if (!this– >healBuffer.loadFromFile("melodies/healing.wav")) {

cout << " ! ERROR::GAME::INITSOUNDS::COULD NOT LOAD HELING SOUND\n";

}

if (!this– >damagBuffer.loadFromFile("melodies/damage.flac")) {

cout << " ! ERROR::GAME::INITSOUNDS::COULD NOT LOAD DAMAGE SOUND\n";

}

if (!this– >coinBuffer.loadFromFile("melodies/coins.flac")) {

cout << " ! ERROR::GAME::INITSOUNDS::COULD NOT LOAD COINS SOUND\n";

}

if (!this– >deathBuffer.loadFromFile("melodies/death.wav")) {

cout << " ! ERROR::GAME::INITSOUNDS::COULD NOT LOAD DEATH SOUND\n";

}

if (!this– >restartBuffer.loadFromFile("melodies/button\_click.ogg")) {

cout << " ! ERROR::GAME::INITSOUNDS::COULD NOT LOAD BUTTON CLICK SOUND\n";

}

if (!this– >recordBuffer.loadFromFile("melodies/newRecord.wav")) {

cout << " ! ERROR::GAME::INITSOUNDS::COULD NOT LOAD RECORD SOUND\n";

}

this– >healSound.setBuffer(this– >healBuffer);

this– >healSound.setVolume(50.f);

this– >damagSound.setBuffer(this– >damagBuffer);

this– >damagSound.setVolume(50.f);

this– >coinSound.setBuffer(this– >coinBuffer);

this– >coinSound.setVolume(50.f);

this– >deathSound.setBuffer(this– >deathBuffer);

this– >deathSound.setVolume(50.f);

this– >restartSound.setBuffer(this– >restartBuffer);

this– >restartSound.setVolume(20.f);

this– >recordSound.setBuffer(this– >recordBuffer);

this– >recordSound.setVolume(70.f);

}

//Constructors and Destructors

Game::Game() {

this– >initWindow();

this– >initTexture();

this– >initSprite();

this– >initFonts();

this– >initVariables();

this– >initGUI();

this– >initSounds();

}

Game::~Game() {

delete this– >window;

}

//Gmame loop

void Game::runGame() {

while (this– >running()) {

this– >pollEvents();

if (this– >endGame == true) {

this– >updateMousePosition();

this– >updateUserCollisionWithButtons();

if (this– >isClickOnRestart == true) {

this– >restartSound.play();

this– >window– >close();

Game game;

game.runGame();

}

}

else {

this– >update();

}

this– >render();

}

}

const bool& Game::getEndGame() const {

return this– >endGame;

}

//Functions

const bool Game::running() const {

return this– >window– >isOpen();//&& this– >endGame == false;

}

void Game::pollEvents() {

Event sfmlEvent;

while (this– >window– >pollEvent(sfmlEvent)) {

switch (sfmlEvent.type) {

case (Event::Closed): {

this– >window– >close();

break;

}

case (Event::KeyPressed): {

if (sfmlEvent.key.code == Keyboard::Escape) {

this– >window– >close();

}

if (sfmlEvent.key.code == Keyboard::R) {

this– >window– >close();

Game game;

game.runGame();

}

break;

}

}

}

}

void Game::spawnSwagBalls() {

//Timer

if (this– >spawnTimer < this– >spawnTimerMax) {

this– >spawnTimer += 1.f;

}

else {

if (this– >swagBalls.size() < this– >maxSwagBalls) {

int tempType = this– >randBallType();

if (tempType == SwagBallsTypes::DAMAGING) {

if (this– >amountDamaging < this– >maxAmountDamaging) {

this– >swagBalls.push\_back(new SwagBall(this– >window, tempType));

this– >amountDamaging++;

}

else {

tempType = this– >randBallType();

}

}

else if (tempType == SwagBallsTypes::HEALING) {

if (this– >amountHealling < this– >maxAmountHealling) {

this– >swagBalls.push\_back(new SwagBall(this– >window, tempType));

this– >amountHealling++;

}

else {

tempType = this– >randBallType();

}

}

else if(tempType == SwagBallsTypes::DEFAULT){

this– >swagBalls.push\_back(new SwagBall(this– >window, tempType));

}

this– >spawnTimer = 0.f;

}

}

}

//Set palayer score

void Game::setScore() {

try {

ifstream fin(this– >path);

if (!fin.is\_open()) {

throw "! ERROR::OF::OPENING::FILE::Score.txt\n";

}

fin>> this– >rateScore;

}

catch (const string& str) {

cout << str;

}

}

//Randomize types balls

const int Game::randBallType() {

int type = SwagBallsTypes::DEFAULT;

int randValue = rand() % 100 + 1;

if (randValue >= 60 && randValue <= 80) {

type = SwagBallsTypes::DAMAGING;

}

else if (randValue > 80 && randValue <= 100) {

type = SwagBallsTypes::HEALING;

}

return type;

}

void Game::updateMousePosition() {

//Get mouse position relative window in pixels

this– >mousePositionPixels = Mouse::getPosition(\*this– >window);

//Convert pixels in coordinates x, y

this– >mousePositionCoordinates = this– >window– >mapPixelToCoords(this– >mousePositionPixels);

}

void Game::updateUserCollisionWithButtons() {

if (Mouse::isButtonPressed(Mouse::Left)) {

if (this– >restartButton.getGlobalBounds().contains(this– >mousePositionCoordinates)) {

this– >isClickOnRestart = true;

}

}

}

void Game::updateGUI() {

stringstream score;

score << "Score: " << this– >points << " / "

<< this– >rateScore << "\n";

this– >guiText.setString(score.str());

stringstream health;

health << this– >player.getHP() << " / " << this– >player.getHPMax() << "\n";

this– >healthText.setString(health.str());

//Update player health bar

float healthPercent=static\_cast<float>(this– >player.getHP())/this– >player.getHPMax();

this– >healthBar.setSize(Vector2f(300.f\*healthPercent,this>healthBar.getSize().y));

}

void Game::updateCollision() {

//Check the collision and delete swagBall

for (int i = 0; i < this– >swagBalls.size(); i++) {

FloatRect playerBounds = this– >player.getSprite().getGlobalBounds();

FloatRect enemyBounds = this– >

swagBalls[i]>getShape().getGlobalBounds();

if (playerBounds.intersects(enemyBounds)) {

switch (this– >swagBalls[i]– >getType()) {

case SwagBallsTypes::DEFAULT: {

this– >coinSound.play();

if (this– >points < this– >rateScore) {

this– >points++;

}

else {

if (this– >isBeatRecord) {

this– >recordSound.play();

}

this– >isBeatRecord = false;

this– >points++;

this– >rateScore++;

}

break;

}

case SwagBallsTypes::DAMAGING: {

this– >damagSound.play();

this– >player.takeDamage(3);

this– >amountDamaging– – ;

break;

}

case SwagBallsTypes::HEALING: {

this– >healSound.play();

this– >player.gainHealth(2);

this– >amountHealling– – ;

break;

}

}

//Remove the ball

this– >swagBalls.erase(this– >swagBalls.begin() + i);

}

}

}

void Game::updatePlayer() {

this– >player.update(this– >window);

if (this– >player.getHP() <= 0) {

try {

ofstream fout(this– >path);

if (!fout.is\_open()) {

throw "! ERROR::OF::OPENING::FILE::Score.txt\n";

}

fout << this– >rateScore;

}

catch (const string& str) {

cout << str;

}

this– >endGame = true;

this– >deathSound.play();

}

}

void Game::update() {

this– >spawnSwagBalls();

this– >updatePlayer();

this– >updateCollision();

this– >updateGUI();

if (this– >endGame == true) {

this– >updateMousePosition();

this– >updateUserCollisionWithButtons();

}

}

void Game::renderGUIButtons() {

//Render restart button

this– >window– >draw(this– >restartButton);

}

void Game::renderGUI(RenderTarget\* target) {

target– >draw(this– >guiText);

target– >draw(this– >healthBarBack);

target– >draw(this– >healthBar);

target– >draw(this– >healthText);

}

void Game::render() {

//Clear old stuff

this– >window– >clear();

//Render new stuff

//Render player

this– >window– >draw(this– >background);

this– >player.render(this– >window);

//Render each swagBall

for (auto i : this– >swagBalls) {

i– >render(this– >window);

}

//Render text

this– >renderGUI(this– >window);

//Render game over text

if (this– >endGame == true) {

this– >window– >draw(this– >endGameText);

this– >renderGUIButtons();

}

this– >window– >display();

}

# Додаток Г

Реалізація класів гравця та ворога.

Player.h

#pragma once

#include"InterfacePlayer.h"

enum PLAYER\_ANIMATION\_STATES {

IDLE,

MOVING\_LEFT,

MOVING\_RIGHT,

MOVING\_UPWARDS,

MOVING\_DOWNWARDS

};

class Player: public InterfacePlayer{

private:

Sprite shape;

Texture playerTexture;

short animState;

IntRect currentFrame;

Clock animationTimer;

float movementSpeed;

int healPoint;

int healPointMax;

//Private functions

virtual void initTexture();

virtual void initVariables();

virtual void initSprite();

virtual void initAnimation();

public:

//Constructors and Destructors

Player(float x = (1024.f / 2.f – 37.f), float y = (720.f / 2.f – 41.f));

virtual ~Player();

//Accessors

virtual const Sprite& getSprite() const;

virtual const int& getHP() const;

virtual const int& getHPMax() const;

//Functions

virtual void takeDamage(const int damage);

virtual void gainHealth(const int health);

virtual void updateWindowBoundsCollision(const RenderTarget\* target);

//Player's movements

virtual void updateAnimation();

virtual void updateMovement();

virtual void update(const RenderTarget\* target);

virtual void render(RenderTarget\* target);

};

Player.cpp

#include "Player.h"

//Private functions

void Player::initTexture() {

if (!this– >playerTexture.loadFromFile("images/player.png")) {

cout << "ERROR::PLAYER::INITTEXTURE::Could not load texture file.\n";

}

}

void Player::initVariables() {

this– >movementSpeed = 1.5f;

this– >healPointMax = 10;

this– >healPoint = this– >healPointMax;

this– >animState = PLAYER\_ANIMATION\_STATES::IDLE;

}

void Player::initSprite() {

/\*this– >shape.setFillColor(Color::Green);

this– >shape.setSize(Vector2f(50.f, 50.f));\*/

this– >shape.setTexture(this– >playerTexture);

this– >currentFrame = IntRect(48, 5, 46, 48);

this– >shape.setTextureRect(this– >currentFrame);

this– >shape.setScale(Vector2f(1.7, 1.7));

}

void Player::initAnimation() {

this– >animationTimer.restart();

}

//Constructors and Destructors

Player::Player(float x, float y) {

//Setting starting position for player

this– >shape.setPosition(x, y);

this– >initTexture();

this– >initSprite();

this– >initAnimation();

this– >initVariables();

}

Player::~Player() {

}

//Accessors

const Sprite& Player::getSprite() const {

return this– >shape;

}

const int& Player::getHP() const {

return this– >healPoint;

}

const int& Player::getHPMax() const {

return this– >healPointMax;

}

//Functions

void Player::takeDamage(const int damage) {

if (this– >healPoint > 0) {

this– >healPoint – = damage;

}

if (this– >healPoint < 0) {

this– >healPoint = 0;

}

}

void Player::gainHealth(const int health) {

if (this– >healPoint < this– >healPointMax) {

this– >healPoint += health;

}

if (this– >healPoint >= this– >healPointMax) {

this– >healPoint = this– >healPointMax;

}

}

void Player::updateWindowBoundsCollision(const RenderTarget\* target) {

/\*FloatRect has 4 values:

– left: x coordinate

– top: y coordinate

– height / width

\*/

//Left bound collision

if (this– >shape.getGlobalBounds().left + 10.f <= 0.f) {

this– >shape.setPosition(– 10.f, this– >shape.getGlobalBounds().top);

}

//Right bound collision

if (this– >shape.getGlobalBounds().left + this– >shape.getGlobalBounds().width – 11.f >= target– >getSize().x) {

this– >shape.setPosition(target– >getSize().x – this– >shape.getGlobalBounds().width + 11.f, this– >shape.getGlobalBounds().top);

}

//Top bound collision

if (this– >shape.getGlobalBounds().top <= 0.f) {

this– >shape.setPosition(this– >shape.getGlobalBounds().left, 0.f);

}

//Bottom bound collision

if (this– >shape.getGlobalBounds().top + this– >shape.getGlobalBounds().height >= target– >getSize().y) {

this– >shape.setPosition(this– >shape.getGlobalBounds().left, target– >getSize().y – this– >shape.getGlobalBounds().height);

}

}

void Player::updateMovement() {

this– >animState = PLAYER\_ANIMATION\_STATES::IDLE;

//Left movement

if (Keyboard::isKeyPressed(Keyboard::A) || Keyboard::isKeyPressed(Keyboard::Left)) {

this– >shape.move(– this– >movementSpeed, 0.f);

this– >animState = PLAYER\_ANIMATION\_STATES::MOVING\_LEFT;

}

//Right movement

else if (Keyboard::isKeyPressed(Keyboard::D) || Keyboard::isKeyPressed(Keyboard::Right)) {

this– >shape.move(this– >movementSpeed, 0.f);

this– >animState = PLAYER\_ANIMATION\_STATES::MOVING\_RIGHT;

}

//Upwards movement

else if (Keyboard::isKeyPressed(Keyboard::W) || Keyboard::isKeyPressed(Keyboard::Up)) {

this– >shape.move(0.f, – this– >movementSpeed);

this– >animState = PLAYER\_ANIMATION\_STATES::MOVING\_UPWARDS;

}

//Downwards movemnt

else if (Keyboard::isKeyPressed(Keyboard::S) || Keyboard::isKeyPressed(Keyboard::Down)) {

this– >shape.move(0.f, this– >movementSpeed);

this– >animState = PLAYER\_ANIMATION\_STATES::MOVING\_DOWNWARDS;

}

}

void Player::updateAnimation() {

if (this– >healPoint <= 0) {

this– >currentFrame.top = 342;

this– >currentFrame.left = 486;

this– >currentFrame.width = 35;

this– >currentFrame.height = 42;

this– >shape.setTextureRect(this– >currentFrame);

}

else {

if (this– >animState == PLAYER\_ANIMATION\_STATES::MOVING\_DOWNWARDS) {

if (this– >animationTimer.getElapsedTime().asSeconds() >= 0.2f) {

this– >currentFrame.top = 0.f;

this– >currentFrame.left += 48.f;

if (this– >currentFrame.left >= 144.f) {

this– >currentFrame.left = 0;

}

this– >animationTimer.restart();

this– >shape.setTextureRect(this– >currentFrame);

}

}

else if (this– >animState == PLAYER\_ANIMATION\_STATES::MOVING\_LEFT) {

if (this– >animationTimer.getElapsedTime().asSeconds() >= 0.2f) {

this– >currentFrame.top = 48.f;

this– >currentFrame.left += 48.f;

if (this– >currentFrame.left >= 144.f) {

this– >currentFrame.left = 0;

}

this– >animationTimer.restart();

this– >shape.setTextureRect(this– >currentFrame);

}

}

else if (this– >animState == PLAYER\_ANIMATION\_STATES::MOVING\_RIGHT) {

if (this– >animationTimer.getElapsedTime().asSeconds() >= 0.2f) {

this– >currentFrame.top = 98.f;

this– >currentFrame.left += 48.f;

if (this– >currentFrame.left >= 144.f) {

this– >currentFrame.left = 0;

}

this– >animationTimer.restart();

this– >shape.setTextureRect(this– >currentFrame);

}

}

else if (this– >animState == PLAYER\_ANIMATION\_STATES::MOVING\_UPWARDS) {

if (this– >animationTimer.getElapsedTime().asSeconds() >= 0.2f) {

this– >currentFrame.top = 148.f;

this– >currentFrame.left += 48.f;

if (this– >currentFrame.left >= 144.f) {

this– >currentFrame.left = 0;

}

this– >animationTimer.restart();

this– >shape.setTextureRect(this– >currentFrame);

}

}

}

}

void Player::update(const RenderTarget\* target) {

//Keyboard input

this– >updateMovement();

this– >updateAnimation();

//Window bounds collision

this– >updateWindowBoundsCollision(target);

}

void Player::render(RenderTarget\* target) {

target– >draw(this– >shape);

}

SwagBall.h

#pragma once

#include "InterfaceEnemy.h"

enum SwagBallsTypes {DEFAULT = 0, DAMAGING, HEALING, NROFTYPES};

class SwagBall: public InterfaceEnemy {

private:

Sprite shape;

Texture damagingTexture;

Texture heallingTexture;

Texture defaultTexture;

//CircleShape shape;

int type;

//Private functions

virtual void initShape(const RenderWindow\* window);

public:

//Constructors and Destructors

SwagBall(const RenderWindow\* window, int type);

virtual ~SwagBall();

//Accessors

virtual const Sprite getShape() const;

virtual const int& getType() const;

//Functions

virtual void render(RenderTarget\* target);

};

SwagBall.cpp

#include "SwagBall.h"

//Private functions

void SwagBall::initShape(const RenderWindow\* window) {

switch (this– >type) {

case SwagBallsTypes::DEFAULT: {

this– >defaultTexture.loadFromFile("images/coin.png");

this– >shape.setTexture(this– >defaultTexture);

this– >shape.setTextureRect(IntRect(0, 0, 105, 105));

this– >shape.setScale(Vector2f(0.3, 0.3));

break;

}

case SwagBallsTypes::DAMAGING: {

this– >damagingTexture.loadFromFile("images/tnt.png");

this– >shape.setTexture(this– >damagingTexture);

this– >shape.setTextureRect(IntRect(39, 41, 360, 360));

this– >shape.setScale(Vector2f(0.12, 0.12));

break;

}

case SwagBallsTypes::HEALING: {

this– >heallingTexture.loadFromFile("images/heart.png");

this– >shape.setTexture(this– >heallingTexture);

this– >shape.setTextureRect(IntRect(0, 4, 110, 105));

this– >shape.setScale(Vector2f(0.4, 0.4));

break;

}

}

float xCoord = static\_cast<float>(

rand() % window– >getSize().x – this– >shape.getGlobalBounds().width

);

float yCoord = static\_cast<float>(

rand() % window– >getSize().y – this– >shape.getGlobalBounds().height

);

//Check out wether coordinates spawn enemy is in window

//Check top and bottom bounds

if(yCoord <= 0 | yCoord >= window– >getSize().y) {

yCoord = static\_cast<float>(

rand() % window– >getSize().y – this– >shape.getGlobalBounds().height

);

}

//Check right and left bounds

if (xCoord <= 0 | xCoord >= window– >getSize().x) {

xCoord = static\_cast<float>(

rand() % window– >getSize().x – this– >shape.getGlobalBounds().width

);

}

Vector2f position(xCoord, yCoord);

this– >shape.setPosition(position);

}

//Constructors and Destructors

SwagBall::SwagBall(const RenderWindow\* window, int type) : type(type) {

this– >initShape(window);

}

SwagBall::~SwagBall() {

}

const Sprite SwagBall::getShape() const {

return (this– >shape);

}

const int& SwagBall::getType() const {

return this– >type;

}

//Functions

void SwagBall::render(RenderTarget\* target) {

target– >draw(this– >shape);

}

# Додаток Д:

Заголовковий файл для підключення бібліотек, які використанні у написанні програми.

#pragma once

#include<SFML/Graphics.hpp>

#include<SFML/System.hpp>

#include<SFML/Window.hpp>

#include<SFML/Audio.hpp>

#include<SFML/Network.hpp>

#include<iostream>

#include<vector>

#include<ctime>

#include<sstream>

#include<fstream>

using namespace sf;

using namespace std;

# Додаток Є:

Реалізація інтерфейсів для опису поводження ігрових об’єктів класів.

InterfacePlayer.h

#pragma once

#include "libraries.h"

class InterfacePlayer {

public:

virtual void initTexture() = 0;

virtual void initVariables() = 0;

virtual void initSprite() = 0;

virtual void initAnimation() = 0;

//Accessors

virtual const Sprite& getSprite() const = 0;

virtual const int& getHP() const = 0;

virtual const int& getHPMax() const = 0;

//Functions

virtual void takeDamage(const int damage) = 0;

virtual void gainHealth(const int health) = 0;

virtual void updateWindowBoundsCollision(const RenderTarget\* target) = 0;

virtual void updateAnimation() = 0;

virtual void updateMovement() = 0;

virtual void update(const RenderTarget\* target) = 0;

virtual void render(RenderTarget\* target) = 0;

};

InterfaceEnemy.h

#pragma once

#include "libraries.h"

class InterfaceEnemy {

public:

virtual void initShape(const RenderWindow\* window) = 0;

//Accessors

virtual const Sprite getShape() const = 0;

virtual const int& getType() const = 0;

//Functions

virtual void render(RenderTarget\* target) = 0;

};