ДНІПРОВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ОЛЕСЯ ГОНЧАРА

Факультет прикладної математики та інформаційних технологій

Кафедра інженерії програмного забезпечення та інформаційних технологій

**КУРСОВА РОБОТА**

з дисципліни «Об’єктно-орієнтоване програмування»

на тему: «Наслідування, поліморфізм та об’єктно-орієнтований дизайн»

Студента\_1\_курсу групи \_ПЗ-24у-1\_\_\_\_\_

Спеціальності 121 Інженерія програмного

забезпечення

Кондрачук О.В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Керівник \_\_Земляний О.Д.\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість білів \_\_\_\_\_\_\_\_\_\_\_\_

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_

Члени комісії: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_Антоненко С.В.\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_Полонська А.Є.\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_Земляний О.Д.\_\_\_\_\_

(підпис) (прізвище та ініціали)

м. Дніпро, 2024 р.

**Зміст**

[Вступ 2](#_Toc186040745)

[1. Постановка задачі 3](#_Toc186040746)

[2. Теоретична частина та початкові налаштування 4](#_Toc186040747)

[2.1 Детальний опис завдань до курсової роботи 4](#_Toc186040748)

[2.2 Основні принципи ООП 5](#_Toc186040749)

[2.3 Патерн проєктування Композит 6](#_Toc186040750)

[2.4 Принципи об'єктно-орієнтованого дизайну 7](#_Toc186040751)

[3. Практична частина 8](#_Toc186040752)

[3.1 Встановлення SFML в середовище Visual Studio 2022 9](#_Toc186040753)

[4. Опис інтерфейсу 13](#_Toc186040754)

[5. Опис тестових прикладів 34](#_Toc186040755)

[6. Перевірка звіту на плагіат 42](#_Toc186040756)

# **Вступ**

Ця курсова робота присвячена створенню програмного забезпечення для роботи з графічними об'єктами у двовимірному просторі. Головна мета проєкту — розробити систему, яка дозволяє створювати геометричні фігури, деформувати, переміщувати їх, змінювати колір. Також система підтримує можливість об'єднання простих елементів у один композит, що дозволяє будувати біль складні структури. При цьому складні об'єкти зберігають усі можливості базових елементів.

Проєкт реалізовано з урахуванням ключових принципів об'єктно-орієнтованого програмування: інкапсуляції, наслідування та поліморфізму. У теоретичній частині аналізується застосування патерну "Композит", що дозволяє легко працювати зі складними об'єктами. Також у роботі розглядаються основи об'єктно-орієнтованого дизайну, які сприяють написанню зрозумілого, чистого та підтримуваного коду.

Практична частина містить покрокову інструкцію з налаштування бібліотеки SFML у середовищі розробки Visual Studio 2022. Також, наводиться тестовий приклад програми, який демонструє основні можливості розробленої системи.

1. **Постановка задачі**

Скласти програму, яка в діалоговому режимі керує графічними об’єктами, що відображаються на екрані дисплея.

Програма повинна підтримувати такі загальні елементи поведінки графічних об’єктів:

1. Активізація/візуалізація за вибором.
2. Переміщення зі слідом/без.
3. Відновлення початкового стану образа.
4. Зміна кольору.
5. Зміна стану видимий/невидимий.
6. При агрегації об’єктів забезпечити можливість агрегації агрегатів.

Програма повинна бути розроблена згідно вимогам об’єктно-орієнтованого дизайну та угодам з написання коду. Важливо, щоб програма не містила «магічних» констант, коду, що дублюється, витоків пам'яті (тобто кожному new повинно відповідати свій delete у потрібній формі).

Звіт до роботи оформлюється звичайним чином, але з титульним листом курсової роботи. Після узгодження електронного варіанту звіту потрібно буде надати паперовий екземпляр. Передбачено захист роботи перед комісією.

Індивідуальний варіант завдань

Варіанти поведінки об’єктів.

1. Завершення роботи з об’єктом: Залишається на екрані.

2. Рух об’єкта в автоматичному режимі: По заданому закону.

3. Зміна кольору і деформація: Зміна кольору по команді, деформація під впливом іншого об’єкта.

4. Дія при досягненні краю вікна: Рух продовжується з протилежного краю вікна.

5. Зборка/запам’ятання агрегованого образа: Видаленням.

Номер індивідуального варіанта представляє собою послідовність нулів та одиниць. Вибір варіанта потрібно виконувати в залежності від особистого номера студента N в журналі академгрупи наступним чином: поставимо у відповідність i-му пункту завдання (5-i)-ю степінь двійки в двоічному представленні номера N. Номера підпунктів завдання будемо вважати коефіцієнтами розкладення N за степенями двійки. Таким чином, наприклад, номеру N=12 відповідає номер індивідуального завдання 01100 (8+4), тобто пункти 1.0, 2.1, 3.1, 4.0, 5.0.

Варіанти графічних образів пропонується згенерувати самостійно на основі таких графічних примітивів: точка, відрізок, трикутник, прямокутник, коло, зірка. Наприклад: кільце; зірка, вписана в коло; коло, вписане в трикутник, сузір’я і т.ін.

1. **Теоретична частина та початкові налаштування**
2. Детальний опис завдань до курсової роботи

*Активізація/візуалізація за вибором* – функціонал додатку повинен надавати можливість переключатися між об’єктами на полі. Згідно варіанту завдання користувач повинен бачити всі фігури на полі або лише активну.

*Переміщення зі слідом/без* – функціонал додатку повинен надавати можливість міняти положення об’єктів на полі, та за вибором оставляти слід, тобто не затирати минуле положення на екрані.

*Відновлення початкового стану образа* – функціонал додатку повинен надавати можливість після будь-яких дій над об’єктом, повертати його у початковий стан.

*Зміна стану видимий/невидимий* – якщо обрано, що об’єкт невидимий, колір об’єкту та поля повинні стати однаковими, тобто об’єкт малюється кольором фону.

*Зміна кольору* – функціонал додатку повинен надавати можливість міняти колір об’єктів.

*Деформація об’єкта* – функціонал додатку повинен надавати можливість міняти форму об’єктів. Допускається розуміти під цим зміну форми, розміру, миготіння, або інший варіант, який можна запропонувати викладачу.

*Зміна кольору або деформація за командою* – функціонал, коли задана дія відбувається за натисканням якоїсь клавіші, передбаченої інтерфейсом.

*Зміна кольору або деформація під впливом іншого об’єкта* – функціонал, коли задана дія відбувається в результаті зіткнення об’єктів.

*При агрегації об’єктів забезпечити можливість агрегації агрегатів* – в цьому пункті завдання мається на увазі наступне. Прості об’єкти під час роботи програми можуть об’єднуватись у складні фігури, тим самим отримувати нові складні об’єкти, які в свою чергу теж можуть виступати частинами нових складних об’єктів. Зі складними об’єктами повинні виконуватись ті ж самі дії, що і з простими. Цей пункт реалізується за рахунок патерну проєктування Композит.

*Агрегація з видаленням* – мається на увазі, що при додаванні фігури в агрегат вона зникає з колекції фігур контролера (сцени) і стає частиною створюваного агрегату, що потім буде доданий як новий об’єкт до колекції контролера.

*Агрегація з дублюванням* – мається на увазі, що при додаванні фігури в агрегат вона НЕ зникає з колекції фігур контролера (сцени), а її копія стає частиною створюваного агрегату, що потім буде доданий як новий об’єкт до колекції контролера.

1. Основні принципи ООП

Абстракція – це обмежений набір ознак та поведінки, які однозначно ідентифікують сутність у межах в рамках конкретної задачі.

Інкапсуляція – це властивість, що дозволяє об'єднати дані та методи, що працюють з ними, в одній сутності – класі, та приховати деталі реалізації від користувача завдяки модифікаторам доступу до членів класу.

Спадкування – це властивість, що дозволяє описати новий клас на основі вже існуючого класу. Клас, від якого здійснюється спадкування, називається базовим чи батьківським. Новий клас називається нащадком, спадкоємцем чи похідним класом. Новий клас спадкує все, що є у базовому класі, та додає нові дані та методи або перевизначає те, що отримано за наслідуванням.

Поліморфізм – це здатність системи працювати з об'єктами через спільний інтерфейс, не маючи потреби знати їхній конкретний тип або внутрішню структуру. Це дозволяє використовувати різні реалізації одного й того самого інтерфейсу, забезпечуючи універсальність і гнучкість роботи з об'єктами.

Наприклад, припустимо, що ви зчитуєте дані з файлу. Для цього у класі, який реалізує роботу з файловим потоком, є метод byte[] readBytes(int n). Якщо згодом виникає потреба отримувати ті ж самі дані через мережеве підключення, у класі, що працює із сокетами, також буде реалізований метод readBytes(). Завдяки поліморфізму ви можете замінити об'єкт, який працює з файлом, на об'єкт, що використовує сокет, без змін у логіці програми. Система продовжить функціонувати незалежно від джерела даних — чи це локальний файл, чи дані, отримані через мережу.

Цей підхід дозволяє абстрагуватися від деталей конкретної реалізації і працювати лише на рівні інтерфейсу. Єдиною вимогою залишається наявність у кожного об'єкта методу readBytes(), який визначений у спільному інтерфейсі.

1. Патерн проєктування Композит

Патерн Композит дозволяє організувати об'єкти в ієрархічну структуру, де можна працювати з групою об'єктів так само, як із одним. Він об'єднує прості елементи та складені об'єкти через спільний інтерфейс, що забезпечує уніфіковану взаємодію з ними.

Головна ідея патерну полягає в абстрагуванні клієнтського коду від деталей реалізації. Незалежно від того, чи це окремий елемент, чи ціла група, взаємодія відбувається через один інтерфейс. Це полегшує створення складних структур, наприклад, деревоподібних, де кожен вузол може містити дочірні елементи.

Композит широко використовується у додатках, де потрібна ієрархія, наприклад, у графічних редакторах, файлових системах або системах управління користувачами. Він забезпечує гнучкість, дозволяючи змінювати структуру без впливу на клієнтський код. Однак його застосування виправдане лише в ситуаціях, де ієрархія дійсно складна й обґрунтована.

1. Принципи об'єктно-орієнтованого дизайну

Об'єктно-орієнтований дизайн базується на ряді принципів, які спрямовані на створення програмного забезпечення, що легко підтримується, розширюється та адаптується до змін. Ці принципи допомагають організувати код таким чином, щоб він був гнучким, зрозумілим і не містив зайвих залежностей або дублювання.

Одним із ключових аспектів є те, що кожен компонент системи має виконувати лише одну конкретну функцію. Це дозволяє уникнути складності в проєктуванні та спрощує розуміння сутності класів. Якщо об'єкти відповідають лише за одну задачу, їх легше тестувати, змінювати та інтегрувати в більшу систему. При цьому важливо враховувати, що код має бути відкритим для розширення, але закритим для змін. Це означає, що під час додавання нової функціональності не слід змінювати існуючий код, щоб уникнути виникнення помилок у вже протестованих модулях.

Розробка інтерфейсів у системі має відповідати принципу вузької спеціалізації. Не варто створювати об'ємні інтерфейси з великою кількістю методів, оскільки це може призвести до перевантаження клієнтських класів. Краще створювати кілька невеликих і чітко визначених інтерфейсів, що дозволяють забезпечити гнучкість у розробці.

Ще одним важливим принципом є можливість заміни одного об'єкта іншим, якщо вони реалізують однаковий інтерфейс. Це гарантує, що клієнтський код може використовувати будь-який із сумісних об'єктів, не змінюючи власну логіку роботи. Такий підхід дозволяє легко адаптувати систему до нових умов або розширювати її функціональність без значних змін.

Дизайн має також передбачати розподіл залежностей між компонентами системи. Високорівневі модулі не повинні залежати від деталей реалізації низькорівневих модулів. Уся система має базуватися на абстракціях, а конкретні реалізації повинні підлаштовуватися під ці абстракції. Це сприяє зниженню зв'язності між компонентами та забезпечує більшу гнучкість.

Крім того, у процесі розробки важливо дотримуватися простоти. Зайва складність лише ускладнює підтримку, тому проєкти повинні бути максимально зрозумілими як для авторів коду, так і для інших розробників. Уникаючи дублювання коду, можна значно спростити процес внесення змін, адже всі необхідні виправлення потрібно робити лише в одному місці.

Нарешті, варто уникати створення функціональності «про всяк випадок». Часто такі рішення не виправдовують себе, але при цьому збільшують складність проєкту. Програмне забезпечення має розвиватися поступово, відповідно до реальних вимог і потреб.

Застосування цих принципів дозволяє створювати ефективні, надійні та масштабовані системи, що відповідають сучасним стандартам розробки.

1. **Практична частина**

Для роботи з графічними об’єктами пропонується використовувати бібліотеку SFML (Simple and Fast Multimedia Library). Це кросплатформна мультимедійна бібліотека у вільному доступі, що написана мовою С++ та доступна для багатьох відомих мов програмування. З її допомогою можна створювати будь-яку 2D-графіку: від простих геометричних фігур до складних ігор жанру платформер. Вихідний код надається за ліцензією zlib/png, офіційний сайт розробника [www.sfml-dev.org](http://www.sfml-dev.org).

1. Встановлення SFML в середовище Visual Studio 2022

Відкриваємо офіційний сайт розробника <https://www.sfml-dev.org/download/sfml/2.6.2/>, та встановлюємо останню версію SFML, дивись рисунок 3.1

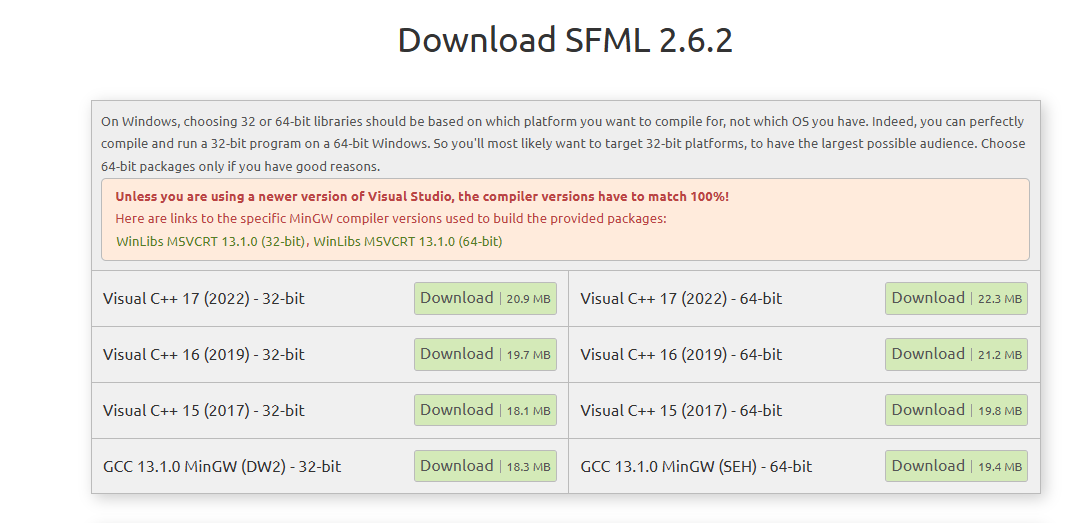


Рисунок 3.1 – Встановлення останньої версії SFML

Далі нам на другому кроці потрібно зайти в Проект–>Cвойства проекту, дивись рисунок 3.2, та відкрити Загальні–>Додаткові каталоги файлів, що включаються та туди додаємо шлях до включених файлів, дивись рисунок 3.3

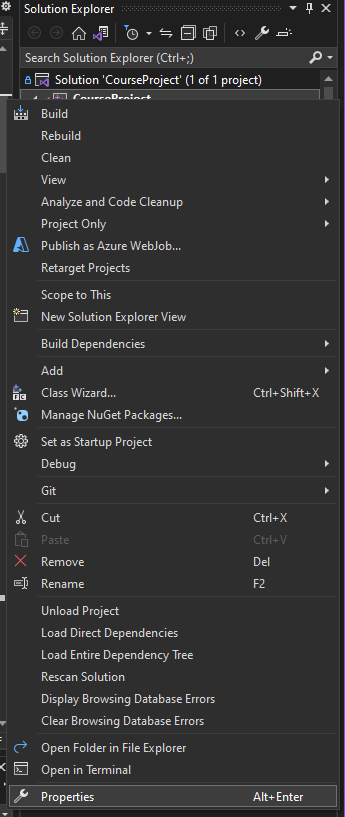


Рисунок 3.2 – Проект–>Cвойства проекту

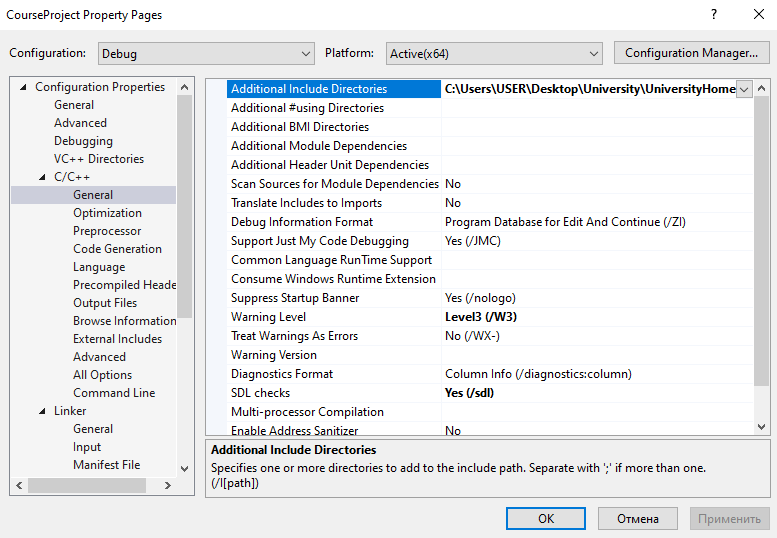


Рисунок 3.3 – Загальні–>Додаткові каталоги файлів, що включаються

Після цього потрібно зайти в Компоновщик–>Додаткові каталоги бібліотек та додати включені бібліотеки з папки SFML яку ми встановили, дивись рисунок 3.4

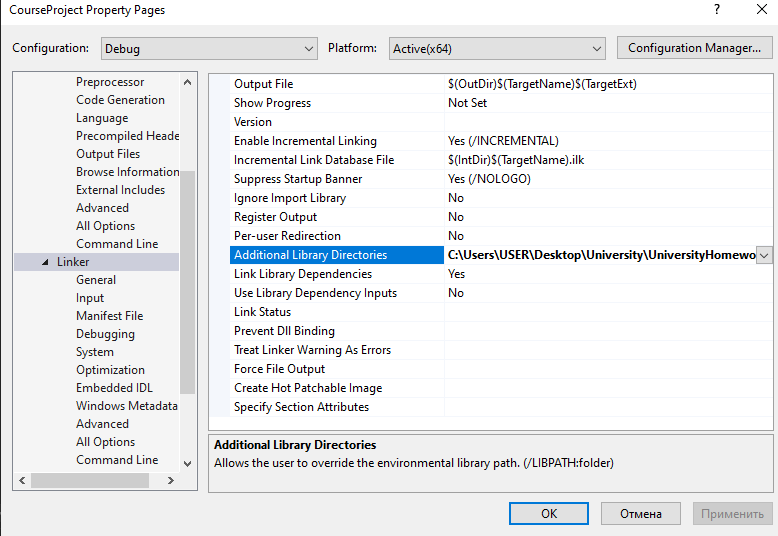


Рисунок 3.4 – Компоновщик–>Додаткові каталоги бібліотек

Далі потрібно зайти в Компоновщик–>Ввод, та додати додаткові залежності, дивись рисунок 3.5

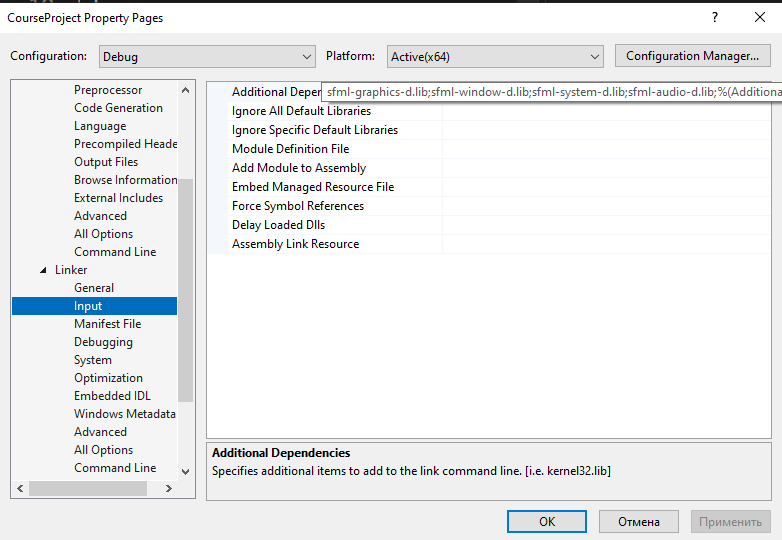


Рисунок 3.5 – Компоновщик–>Ввод, та додавання додаткових залежностей

Також для того щоб SFML повністю працював, потрібно в файл проекта додати бібліотеки, дивись рисунок 3.6

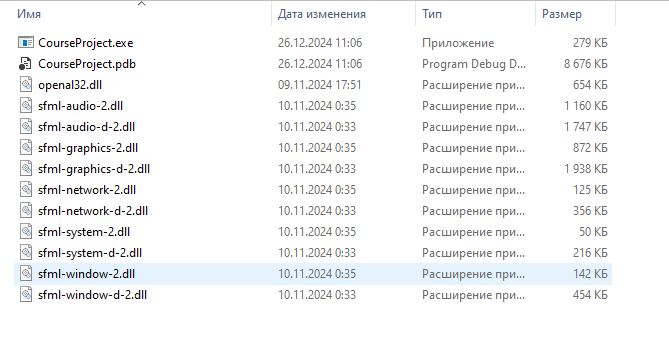


Рисунок 3.6 – Додавання бібліотек до проекту

1. **Опис інтерфейсу**

Лістинг 4.1 – Клас ColorChangeable.h

#pragma once

#include "Input.h"

#include "SelectableUser.h"

class ColorChangeable : public SelectableUser

{

Graphic\* graphic\_;

bool canChange\_;

void awake() override

{

SelectableUser::awake();

canChange\_ = true;

graphic\_ = gameObject->get\_component<Graphic>();

}

void update() override

{

if (!is\_selected())

{

return;

}

if (Input::get\_key(sf::Keyboard::Num2))

{

if (!canChange\_)

{

return;

}

graphic\_->set\_random\_color();

canChange\_ = false;

}

else

{

canChange\_ = true;

}

}

};

Лістинг 4.2 – Клас CompositCreator.h

#pragma once

#include "Input.h"

#include "SelectablesContainer.h"

class CompositeCreator : public MonoBehaviour

{

SelectablesContainer\* container\_;

bool canSelect\_;

void awake() override

{

canSelect\_ = true;

container\_ = gameObject->get\_component<SelectablesContainer>();

}

void update() override

{

if (!Input::get\_key(sf::Keyboard::Num8))

{

container\_->can\_select(true);

return;

}

container\_->can\_select(false);

if (container\_->last\_selected == nullptr)

{

return;

}

if (!Input::get\_mouse\_button(sf::Mouse::Left))

{

canSelect\_ = true;

return;

}

if (!canSelect\_)

{

return;

}

canSelect\_ = false;

SelectableObject\* selectable = container\_->calculate\_selected();

selectable->transform->set\_parent(container\_->last\_selected->transform);

selectable->highlight();

}

};

Лістинг 4.3 – Клас FiguresCreator.h

#pragma once

#include "ColorChangeable.h"

#include "GameObject.h"

#include "GameObjectsContainer.h"

#include "Input.h"

#include "MonoBehaviour.h"

#include "MouseFollower.h"

#include "MovableObject.h"

#include "Retracter.h"

#include "VisibilitySetter.h"

class FiguresCreator : public MonoBehaviour

{

bool canAdd\_;

void awake() override

{

canAdd\_ = true;

}

void update() override

{

if (Input::get\_key(sf::Keyboard::Num0))

{

if (!canAdd\_)

{

return;

}

canAdd\_ = false;

create\_object();

}

else

{

canAdd\_ = true;

}

}

void create\_object()

{

GameObject\* object = new GameObject();

object->add\_component(new SelectableObject());

object->add\_component(new MovableObject());

object->add\_component(new Graphic());

object->add\_component(new ColorChangeable());

object->add\_component(new VisibilitySetter());

object->add\_component(new MouseFollower());

object->add\_component(new Retracter());

GameObject::objectsContainer\_->instantiate(object);

}

};

Лістинг 4.4 – Клас GameObject.h

#pragma once

#include "MonoBehaviour.h"

#include "Transform.h"

class GameObjectsContainer;

class GameObject

{

public:

static GameObjectsContainer\* objectsContainer\_;

bool awakened\_ = false;

std::vector<MonoBehaviour\*> components\_;

public:

Transform\* transform;

GameObject(GameObjectsContainer\* objectsContainer)

{

objectsContainer\_ = objectsContainer;

transform = new Transform(this);

}

GameObject()

{

transform = new Transform(this);

}

virtual ~GameObject()

{

delete transform;

for (const auto component : components\_)

{

delete component;

}

}

void add\_component(MonoBehaviour\* component)

{

components\_.push\_back(component);

component->AddData(this, transform);

if (awakened\_)

{

component->awake();

}

}

void remove\_component(MonoBehaviour\* component)

{

components\_.erase(std::remove(components\_.begin(), components\_.end(), component), components\_.end());

delete component;

}

template <typename T>

T\* get\_component()

{

for (auto\* component : components\_)

{

if (T\* instance = dynamic\_cast<T\*>(component))

{

return instance;

}

}

return nullptr;

}

void awake()

{

awakened\_ = true;

for (const auto component : components\_)

{

component->awake();

}

}

void update() const

{

for (const auto component : components\_)

{

component->update();

}

}

void on\_destroy() const

{

for (const auto component : components\_)

{

component->on\_destroy();

}

}

virtual void on\_collision\_stay(GameObject\* other)

{

for (const auto component : components\_)

{

component->on\_collision\_stay(other);

}

}

};

Лістинг 4.5 – Клас GameObjectsContainer.h

#pragma once

#include <vector>

#include "GameObject.h"

class GameObjectsContainer

{

private:

void check\_for\_collision() const

{

for (int i = 0; i < objects\_.size(); i++)

{

auto object = objects\_[i];

Graphic\* graphic = object->get\_component<Graphic>();

if (graphic == nullptr)

{

continue;

}

for (int j = i + 1; j < objects\_.size(); j++)

{

auto other = objects\_[j];

Graphic\* otherGraphic = other->get\_component<Graphic>();

if (otherGraphic == nullptr)

{

continue;

}

if (graphic->collides(otherGraphic->get\_rect()))

{

object->on\_collision\_stay(other);

other->on\_collision\_stay(object);

}

}

}

}

public:

std::vector<GameObject\*> objects\_;

std::vector<GameObject\*> to\_add\_;

GameObjectsContainer() = default;

void instantiate(GameObject\* object, Transform\* parent = nullptr)

{

to\_add\_.push\_back(object);

object->transform->set\_parent(parent);

object->awake();

}

void destroy(GameObject\* object)

{

objects\_.erase(std::remove(objects\_.begin(), objects\_.end(), object), objects\_.end());

object->transform->set\_parent(nullptr);

object->on\_destroy();

delete object;

}

template<typename T>

std::vector<T\*> find\_of\_type()

{

std::vector<T\*> foundObjects;

for (auto\* object : objects\_) {

if (T\* component = object->get\_component<T>()) {

foundObjects.push\_back(component);

}

}

return foundObjects;

}

void update(sf::RenderWindow\* window)

{

for (auto object : objects\_)

{

object->update();

}

check\_for\_collision();

for (auto element : to\_add\_)

{

objects\_.push\_back(element);

}

to\_add\_.clear();

}

};

Лістинг 4.6 – Клас Graphic.h

#pragma once

#include <SFML/Graphics/CircleShape.hpp>

#include <SFML/Graphics/RectangleShape.hpp>

#include "MonoBehaviour.h"

#include "GameObject.h"

#include "Screen.h"

#include "Utils.h"

enum GraphicForm

{

none = 0,

circle = 1,

rectangle = 2

};

class Graphic : public MonoBehaviour

{

private:

sf::Shape\* shape\_;

sf::Color color\_;

float outlineThickness\_;

bool visible\_;

GraphicForm form\_;

int framesWithoutCollision\_;

void awake() override

{

visible\_ = true;

set\_form(circle);

set\_color(sf::Color::Green);

}

void update() override

{

framesWithoutCollision\_++;

if (!visible\_)

{

return;

}

Screen::get\_window()->draw(\*shape\_, \*transform);

}

void on\_collision\_stay(GameObject\* other) override

{

if (framesWithoutCollision\_ > 1)

{

set\_form(static\_cast<GraphicForm>(form\_ % 2 + 1));

}

framesWithoutCollision\_ = 0;

}

void on\_destroy() override

{

delete shape\_;

}

sf::Shape\* get\_shape() const

{

return shape\_;

}

public:

sf::FloatRect get\_rect() const

{

sf::FloatRect rect = shape\_->getGlobalBounds();

rect.left = transform->get\_position().x;

rect.top = transform->get\_position().y;

return rect;

}

bool collides(const sf::Vector2f position) const

{

return get\_rect().contains(position.x, position.y);

}

bool collides(sf::FloatRect other) const

{

return get\_rect().intersects(other);

}

void set\_outline(const float thickness)

{

shape\_->setOutlineThickness(thickness);

outlineThickness\_ = thickness;

for (auto child : transform->children\_)

{

Graphic\* graphic = child->gameObject->get\_component<Graphic>();

if (graphic != nullptr)

{

graphic->set\_outline(thickness);

}

}

}

void set\_random\_color()

{

int r = Utils::randInclusive(0, 255);

int g = Utils::randInclusive(0, 255);

int b = Utils::randInclusive(0, 255);

color\_ = sf::Color(r, g, b, 255);

shape\_->setFillColor(color\_);

for (auto child : transform->children\_)

{

Graphic\* graphic = child->gameObject->get\_component<Graphic>();

if (graphic != nullptr)

{

graphic->set\_random\_color();

}

}

}

void set\_color(const sf::Color color)

{

shape\_->setFillColor(color);

color\_ = color;

}

sf::Color get\_color() const

{

return shape\_->getFillColor();

}

void set\_visibility(const bool value)

{

visible\_ = value;

for (auto child : transform->children\_)

{

Graphic\* graphic = child->gameObject->get\_component<Graphic>();

if (graphic != nullptr)

{

graphic->set\_visibility(value);

}

}

}

bool get\_visibility() const

{

return visible\_;

}

void set\_form(GraphicForm form)

{

if (form\_ != none)

{

delete shape\_;

}

form\_ = form;

switch (form) {

case circle:

shape\_ = new sf::CircleShape(20.f);

break;

case rectangle:

shape\_ = new sf::RectangleShape({ 40.0f, 40.0f });

break;

case none:

shape\_ = nullptr;

break;

}

set\_color(color\_);

shape\_->setOutlineThickness(outlineThickness\_);

}

};

Лістинг 4.7 – Клас Input.h

#pragma once

#include <SFML/Graphics/RenderWindow.hpp>

#include <SFML/Window/Event.hpp>

class Input

{

private:

static Input\* instance\_;

sf::RenderWindow\* window\_;

public:

Input(sf::RenderWindow\* window)

{

if (instance\_ != nullptr)

{

return;

}

instance\_ = this;

window\_ = window;

}

static bool get\_key(const sf::Keyboard::Key key)

{

return sf::Keyboard::isKeyPressed(key);

}

static bool get\_mouse\_button(const sf::Mouse::Button button)

{

return sf::Mouse::isButtonPressed(button);

}

static sf::Vector2f get\_mouse\_position()

{

sf::Vector2i mousePosition = sf::Mouse::getPosition(\*instance\_->window\_);

return sf::Vector2f(mousePosition.x, mousePosition.y);

}

};

Лістинг 4.8 – Клас MonoBehaviour.h

#pragma once

#include "Transform.h"

class MonoBehaviour

{

public:

GameObject\* gameObject;

Transform\* transform;

virtual ~MonoBehaviour() = default;

void AddData(GameObject\* gameObject, Transform\* transform)

{

this->gameObject = gameObject;

this->transform = transform;

}

virtual void update() {}

virtual void awake() {}

virtual void on\_destroy() {}

virtual void on\_collision\_stay(GameObject\* other) {}

};

Лістинг 4.9 – Клас MouseFollower.h

#pragma once

#include "SelectableUser.h"

class MouseFollower : public SelectableUser

{

std::vector<sf::Vector2f> trajectory\_;

bool moving;

int trajectoryIndex;

void update() override

{

if (!is\_selected())

{

return;

}

if (moving)

{

move\_by\_trajectory();

return;

}

if (Input::get\_key(sf::Keyboard::Num4))

{

trajectory\_.push\_back(Input::get\_mouse\_position());

}

else

{

if (!trajectory\_.empty())

{

moving = true;

}

}

}

void move\_by\_trajectory()

{

if (trajectoryIndex >= trajectory\_.size())

{

moving = false;

trajectoryIndex = 0;

trajectory\_.clear();

return;

}

transform->set\_position(trajectory\_[trajectoryIndex]);

trajectoryIndex++;

}

};

Лістинг 4.10 – Клас MovableObject.h

#pragma once

#include "SelectableUser.h"

#include "Time.h"

class MovableObject : public SelectableUser

{

private:

float speed\_;

Graphic\* graphic\_;

void awake() override

{

SelectableUser::awake();

graphic\_ = gameObject->get\_component<Graphic>();

speed\_ = 150;

}

void update() override

{

if (!is\_selected())

{

return;

}

move();

}

void move()

{

sf::Vector2f position = transform->get\_position();

sf::Vector2f direction = get\_direction();

sf::Vector2f newPosition = position + direction \* speed\_ \* Time::get\_delta\_time();

newPosition.x = Utils::repeat(newPosition.x, -graphic\_->get\_rect().width, Screen::get\_size().x);

newPosition.y = Utils::repeat(newPosition.y, -graphic\_->get\_rect().height, Screen::get\_size().y);

transform->set\_position(newPosition);

}

sf::Vector2f get\_direction()

{

sf::Vector2f position;

if (Input::get\_key(sf::Keyboard::Key::A))

{

position.x += -1;

}

if (Input::get\_key(sf::Keyboard::Key::D))

{

position.x += 1;

}

if (Input::get\_key(sf::Keyboard::Key::W))

{

position.y += -1;

}

if (Input::get\_key(sf::Keyboard::Key::S))

{

position.y += 1;

}

return position;

}

};

Лістинг 4.11 – Клас Rectracter.h

#pragma once

#include "SelectableUser.h"

#include "Input.h"

class Retracter : public SelectableUser

{

bool canRetract;

sf::Vector2f startPosition\_;

sf::Color startColor\_;

Graphic\* graphic\_;

void awake() override

{

SelectableUser::awake();

canRetract = true;

graphic\_ = gameObject->get\_component<Graphic>();

}

void update() override

{

if (!is\_selected())

{

return;

}

if (Input::get\_key(sf::Keyboard::Num5))

{

if (!canRetract)

{

return;

}

transform->set\_position(startPosition\_);

graphic\_->set\_color(startColor\_);

canRetract = false;

}

else

{

canRetract = true;

}

}

public:

void memorize\_data()

{

startPosition\_ = transform->get\_position();

startColor\_ = graphic\_->get\_color();

}

};

Лістинг 4.12 – Клас Screen.h

#pragma once

#include <iostream>

#include <SFML/Graphics/RenderWindow.hpp>

class Screen

{

private:

static Screen\* instance\_;

sf::RenderWindow\* window\_;

bool clearWindow\_;

public:

Screen(sf::RenderWindow\* window)

{

if (instance\_ != nullptr)

{

return;

}

instance\_ = this;

window\_ = window;

clearWindow\_ = true;

}

void clear() const

{

if (!clearWindow\_)

{

return;

}

window\_->clear();

}

void display() const

{

window\_->display();

}

static sf::RenderWindow\* get\_window()

{

return instance\_->window\_;

}

static sf::Vector2f get\_size()

{

sf::Vector2u windowSize = instance\_->window\_->getSize();

return sf::Vector2f(static\_cast<float>(windowSize.x), static\_cast<float>(windowSize.y));

}

static void set\_window\_clear(bool value)

{

instance\_->clearWindow\_ = value;

}

static bool get\_window\_clear()

{

return instance\_->clearWindow\_;

}

};

Лістинг 4.13 – Клас SelectableObject.h

#pragma once

#include "Graphic.h"

#include "MonoBehaviour.h"

class Retracter;

class SelectableObject : public MonoBehaviour

{

private:

const float selectedThickness = 2;

const float deselectedThickness = 0;

Graphic\* graphic\_;

Retracter\* retracter\_;

bool selected;

void awake() override;

public:

bool can\_select();

void select();

void deselect();

void highlight() const;

void unhighlight() const;

bool is\_selected() const;

};

Лістинг 4.14 – Клас SelectablesContainer.h

#pragma once

#include "GameObject.h"

#include "MonoBehaviour.h"

#include "SelectableObject.h"

#include "GameObjectsContainer.h"

class SelectablesContainer : public MonoBehaviour

{

std::vector<SelectableObject\*> selectables\_;

bool can\_select\_;

void awake() override

{

can\_select\_ = true;

}

void update() override

{

selectables\_ = GameObject::objectsContainer\_->find\_of\_type<SelectableObject>();

if (!can\_select\_)

{

return;

}

if (!Input::get\_mouse\_button(sf::Mouse::Left))

{

return;

}

SelectableObject\* selected = calculate\_selected();

if (selected != nullptr)

{

if (last\_selected != nullptr)

{

last\_selected->deselect();

}

last\_selected = selected;

last\_selected->select();

}

}

public:

SelectableObject\* last\_selected;

SelectablesContainer() : last\_selected(nullptr)

{

}

void can\_select(const bool value)

{

can\_select\_ = value;

}

SelectableObject\* calculate\_selected() const

{

for (auto element : selectables\_)

{

if (element->can\_select())

{

SelectableObject\* toSelect = element;

while (toSelect->transform->parent\_ != nullptr)

{

toSelect = toSelect->transform->parent\_->gameObject->get\_component<SelectableObject>();

}

return toSelect;

}

}

return nullptr;

}

};

Лістинг 4.15 – Клас SelectableUser.h

#pragma once

#include "MonoBehaviour.h"

#include "SelectableObject.h"

class SelectableUser : public MonoBehaviour

{

private:

SelectableObject\* selectableObject;

protected:

void awake() override

{

selectableObject = gameObject->get\_component<SelectableObject>();

}

bool is\_selected() const

{

return selectableObject->is\_selected();

}

};

Лістинг 4.16 – Клас Time.h

#pragma once

#include <SFML/System/Clock.hpp>

class Time

{

private:

static Time\* instance\_;

sf::Clock clock\_;

float elapsedTime\_;

public:

Time(sf::Clock clock)

{

if (instance\_ != nullptr)

{

return;

}

instance\_ = this;

clock\_ = clock;

}

void restart()

{

elapsedTime\_ = clock\_.restart().asSeconds();

}

static float get\_delta\_time()

{

return instance\_->elapsedTime\_;

}

};

Лістинг 4.17 – Клас Time.h

#pragma once

#include "Input.h"

#include "MonoBehaviour.h"

#include "Screen.h"

class TrailCreator : public MonoBehaviour

{

bool canChange\_;

void awake() override

{

canChange\_ = true;

}

void update() override

{

if (Input::get\_key(sf::Keyboard::Num1))

{

if (!canChange\_)

{

return;

}

Screen::set\_window\_clear(!Screen::get\_window\_clear());

canChange\_ = false;

}

else

{

canChange\_ = true;

}

}

};

Лістинг 4.18 – Клас Transform.h

#pragma once

#include <vector>

#include <SFML/Graphics/Transform.hpp>

#include <SFML/System/Vector2.hpp>

class GameObject;

class Transform : public sf::Transform

{

private:

sf::Vector2f position\_;

sf::Vector2f get\_parent\_position() const

{

if (parent\_ == nullptr)

{

return sf::Vector2f(0, 0);

}

return parent\_->get\_position();

}

public:

std::vector<Transform\*> children\_;

Transform\* parent\_;

GameObject\* gameObject;

Transform(GameObject\* gameObject)

{

parent\_ = nullptr;

this->gameObject = gameObject;

}

sf::Vector2f get\_position() const

{

return position\_;

}

void set\_position(sf::Vector2f position)

{

for (auto child : children\_)

{

child->set\_position(position + child->get\_local\_position());

}

sf::Vector2f delta = position - position\_;

translate(delta.x, delta.y);

position\_ = position;

}

sf::Vector2f get\_local\_position() const

{

return position\_ - get\_parent\_position();

}

void set\_local\_position(sf::Vector2f position)

{

set\_position(position + get\_parent\_position());

}

Transform\* get\_parent() const

{

return parent\_;

}

void set\_parent(Transform\* parent)

{

if (parent\_ != nullptr)

{

parent\_->children\_.erase(std::remove(parent\_->children\_.begin(), parent\_->children\_.end(), this), parent\_->children\_.end());

}

parent\_ = parent;

if (parent\_ != nullptr)

{

parent\_->children\_.push\_back(this);

}

}

};

Лістинг 4.19 – Клас Utils.h

#pragma once

#include <random>

class Utils

{

public:

static int randInclusive(int min, int max) {

static std::random\_device rd;

static std::mt19937 gen(rd());

std::uniform\_int\_distribution<> dis(min, max);

return dis(gen);

}

static float repeat(float t, float min, float max)

{

if (t > max)

{

t -= (max - min);

}

else if (t < min)

{

t += (max - min);

}

return t;

}

};

Лістинг 4.20 – Клас VisibilitySetter.h

#pragma once

#include "SelectableUser.h"

class VisibilitySetter : public SelectableUser

{

Graphic\* graphic\_;

bool canChange\_;

void awake() override

{

SelectableUser::awake();

canChange\_ = true;

graphic\_ = gameObject->get\_component<Graphic>();

}

void update() override

{

if (!is\_selected())

{

return;

}

if (Input::get\_key(sf::Keyboard::Num3))

{

if (!canChange\_)

{

return;

}

graphic\_->set\_visibility(!graphic\_->get\_visibility());

canChange\_ = false;

}

else

{

canChange\_ = true;

}

}

};

Лістинг 4.21 – Файл main.cpp

#include <iostream>

#include <SFML/Graphics.hpp>

#include "ColorChangeable.h"

#include "CompositeCreator.h"

#include "FiguresCreator.h"

#include "MonoBehaviour.h"

#include "GameObjectsContainer.h"

#include "Input.h"

#include "MouseFollower.h"

#include "MovableObject.h"

#include "Retracter.h"

#include "Screen.h"

#include "SelectableObject.h"

#include "SelectablesContainer.h"

#include "Time.h"

#include "TrailCreator.h"

#include "VisibilitySetter.h"

void add\_objects(GameObjectsContainer\* container)

{

GameObject\* mainObject = new GameObject();

mainObject->add\_component(new SelectablesContainer());

mainObject->add\_component(new TrailCreator());

mainObject->add\_component(new CompositeCreator());

mainObject->add\_component(new FiguresCreator());

container->instantiate(mainObject);

}

int main()

{

sf::RenderWindow window(sf::VideoMode(800, 800), "SFML works!");

sf::Clock clock;

Input input(&window);

Screen screen(&window);

Time time(clock);

GameObjectsContainer container;

GameObject object(&container);

add\_objects(&container);

while (window.isOpen())

{

screen.clear();

time.restart();

container.update(&window);

screen.display();

}

return 0;

}

Цей код містить реалізацію управління графічними об’єктами в SFML із багатьма функціональними можливостями, такими як переміщення, масштабування, додавання нових об’єктів, анімація, автоматичний рух і обробка зіткнень. Основні моменти:

1. Додавання об’єктів за допомогою клавіші 0. Переключення між об’єктами (ліва кнопка миші), зміна кольору (2).
2. Клавіша 8 створює агрегат із поточних об’єктів.
3. Клавіші-WASD дозволяють вручну переміщувати об’єкти, з обробкою виходу за межі екрана (wrap-around).
4. Клавіша 4 активує рух об’єкта по траєкторії миші.
5. Рамка підсвічує поточний об’єкт. При зіткненні об’єкти змінюють форму.
6. Клавіша 5 ставить об’єкт у початкову координату, колір та форму.
7. Клавіша 1 вмикає та вимикає слід від фігур.
8. Клавіша 3 вмикає та вимикає видимість об’єкта.
9. **Опис тестових прикладів**

Тестовий приклад починається зі запуском графічного вікна для взаємодії з об’єктами, дивись рисунок 5.1.

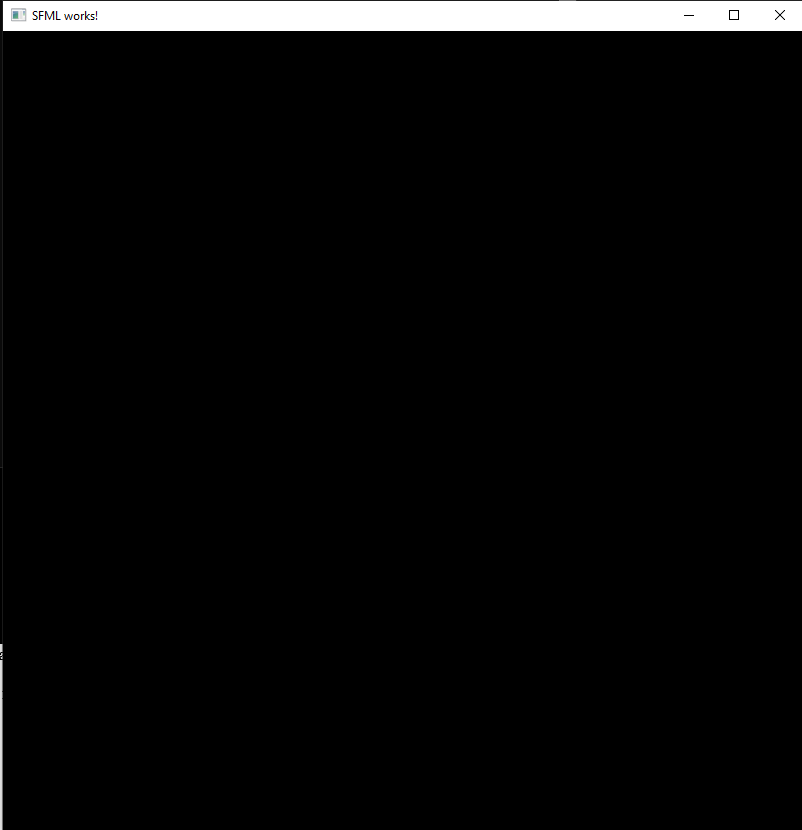


Рисунок 5.1 – Графічне вікно для взаємодії з об’єктами

Далі за допомогою кнопок додаємо наші фігури, дивись рисунок 5.2

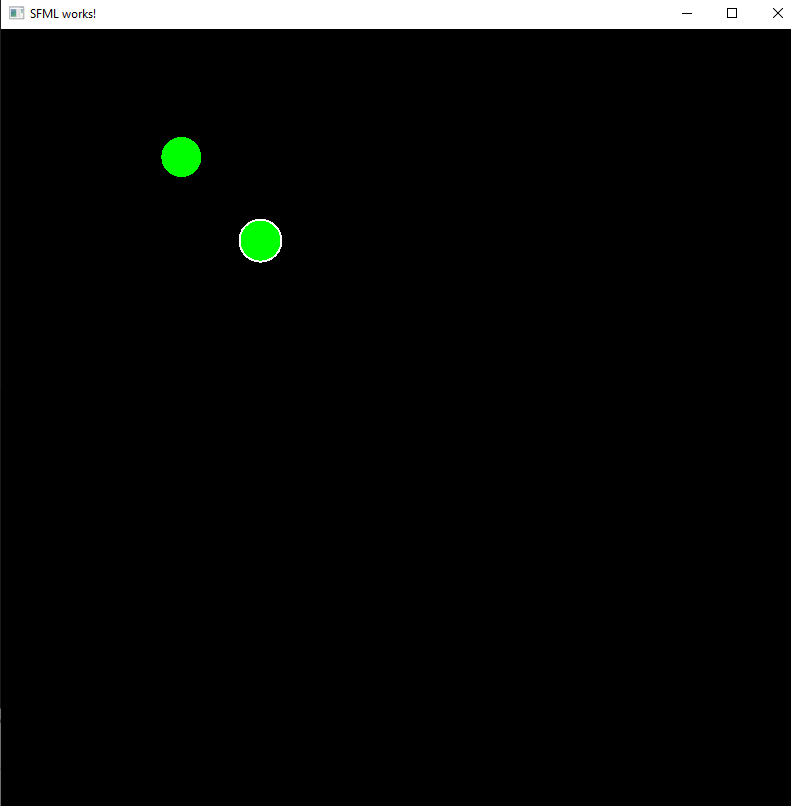


Рисунок 5.2 – Додавання об’єктів

Далі ми можемо перемикатися між цими об’єктами за допомогою клавіш лівої кнопки миші, дивись рисунок 5.3

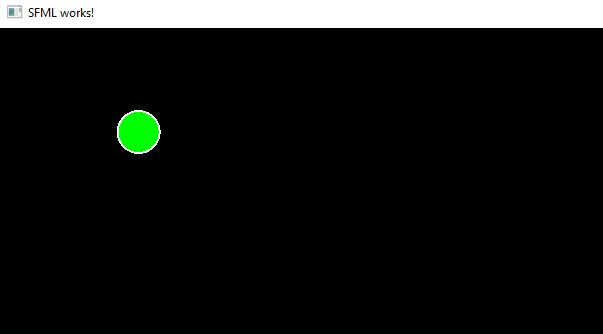


Рисунок 5.3 – Перемикання між об’єктами за допомогою клавіші миші

Потім ми можемо змінити колір поточного об’єкта, дивись рисунок 5.4

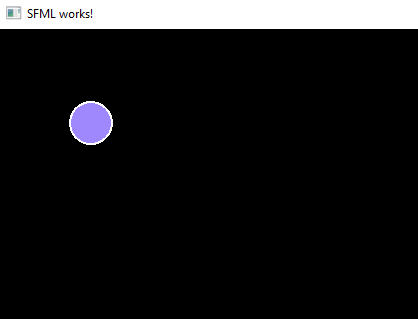


Рисунок 5.4 – Змінення кольору поточного об’єкта

Також ми можемо робити поточний об’єкт видимим чи невидимим, дивись рисунок 5.5



Рисунок 5.5 – Змінення видимості поточного об’єкта

Далі також можливо рухати поточний об’єкт за допомогою стрілочок на клавіатурі, дивись рисунок 5.6



Рисунок 5.6 – Рух поточного об’єкту за допомогою стрілочок

Потім можна розпочати автоматичний рух поточного об’єкта по заданій траєкторії, а задана траєкторія формується за рухом миші, дивись рисунок 5.7.

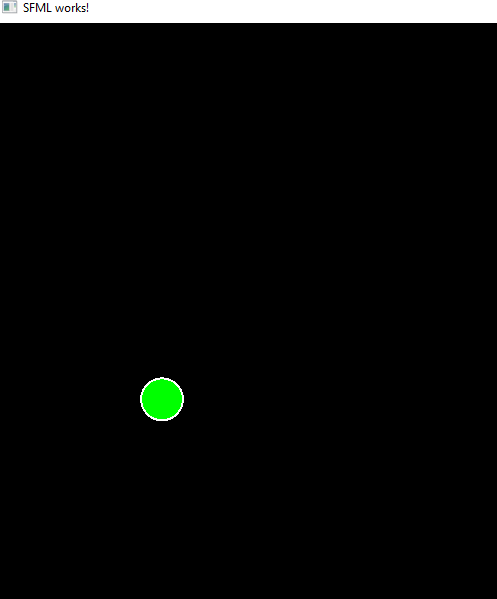


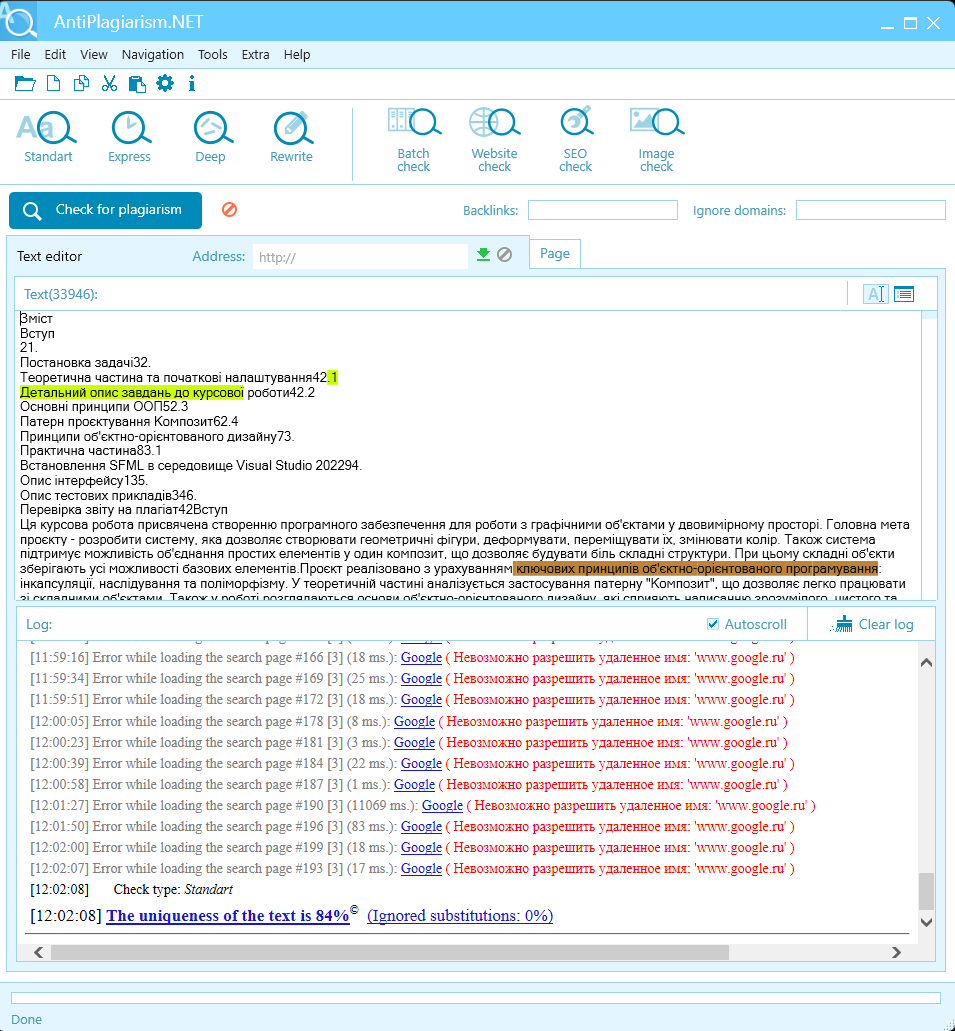
Рисунок 5.7 – Автоматичний рух по заданій траєкторії

Далі в програмі є агрегація об’єктів. За допомогою агрегації, ми создаємо один агрегат де в ньому будуть утримуватися всі ті об’єкти які були на сцені та об’єкти які були на сцені не видаляються, також ми цей агрегат можемо рухати змінити усім об’єктам колір, деформувати і т.д, дивись рисунок 5.9



Рисунок 5.9 – Агрегація об’єктів

1. **Перевірка звіту на плагіат**

****