**Дніпровський національний університет імені ОЛеся Гончара**

**Факультет прикладної математики**

**Кафедра математичного забезпечення ЕОМ**

**КУРСОВА РОБОТА**

з дисципліни «Об’єктно-орієнтоване програмування»

на тему: «Патерни проектування»

Студента 2 курсу групи ПЗ-18-2

Спеціальності 121 Інженерія програмного забезпечення

Волковського В.В.

(прізвище та ініціали)

Керівник

асист. каф. МЗ ЕОМ,

Батурінець А. Г.

Кількість балів \_\_\_\_\_\_\_\_\_\_\_\_

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

Члени комісії : \_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

м. Дніпро, 2020 р.

ЗМІСТ

[1. Постановка задачі 2](#_Toc34972745)

[2. Опис розв’язку 3](#_Toc34972746)

[2.1. Патерн Composite 3](#_Toc34972747)

[2.2. Патерн Prototype 5](#_Toc34972748)

[2.3. Патерн Singleton 6](#_Toc34972749)

[2.4. Патерн Memento 7](#_Toc34972750)

[3. Вихідний текст програми розв’язку задачі 9](#_Toc34972751)

[4. Опис інтерфейсу 10](#_Toc34972752)

[5. Опис тестових прикладів 11](#_Toc34972753)

[6. Аналіз помилок 15](#_Toc34972754)

# 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.

Провести рефакторинг з використанням патернів проектування композит (Composite), прототип (Prototype), одинак (Singleton) і знімок (Memento), таким чином, щоб:

1. агрегат представляв собою патерн композит (Composite);
2. для збирання фігур в агрегат використовувався патерн прототип (Prototype);
3. патерн одинак (Singleton) забезпечував існування лише одного контролера сцени;
4. патерн знімок (Memento) використовувався для зберігання стану наявних фігур у файл на диску.

# Опис розв’язку

Дану роботу реалізовано на основі бібліотеки GLFW.

GLFW – це багатоплатформна бібліотека з відкритим кодом для розробки OpenGL, OpenGL ES та Vulkan на робочому столі. GLFW надає простий API для створення вікон, контекстів, прийому даних та подій.

## Патерн Composite

Призначення патерна composite:

* Необхідно об'єднувати групи схожих об'єктів і управляти ними.
* Об'єкти можуть бути як примітивними (елементарними), так і складовими (складними). Складений об'єкт може включати в себе колекції інших об'єктів, утворюючи складні деревовидні структури. Приклад: директорія файлової системи складається з елементів, кожен з яких також може бути Директорією.
* Код клієнта працює з примітивними і складовими об'єктами одноманітно.

Патерн Composite пропонує наступне рішення. Він вводить інтерфейс з поведінкою, загальним для всіх примітивних і складових об'єктів.

Переваги застосування патерна composite:

* В систему легко додавати нові примітивні або складові об'єкти, так як патерн *Composite* використовує загальний базовий клас *IShape*.
* Код клієнта має просту структуру - примітивні і складові елементи обробляються однаковим чином.
* Патерн Composite дозволяє легко обійти всі вузли деревовидної структури.

Клас *Composite* та клас *IPrimitive* реалізують один інтерфейс (*Ishape*).

Вміст базового класу *IShape* приведено на рис.1 .

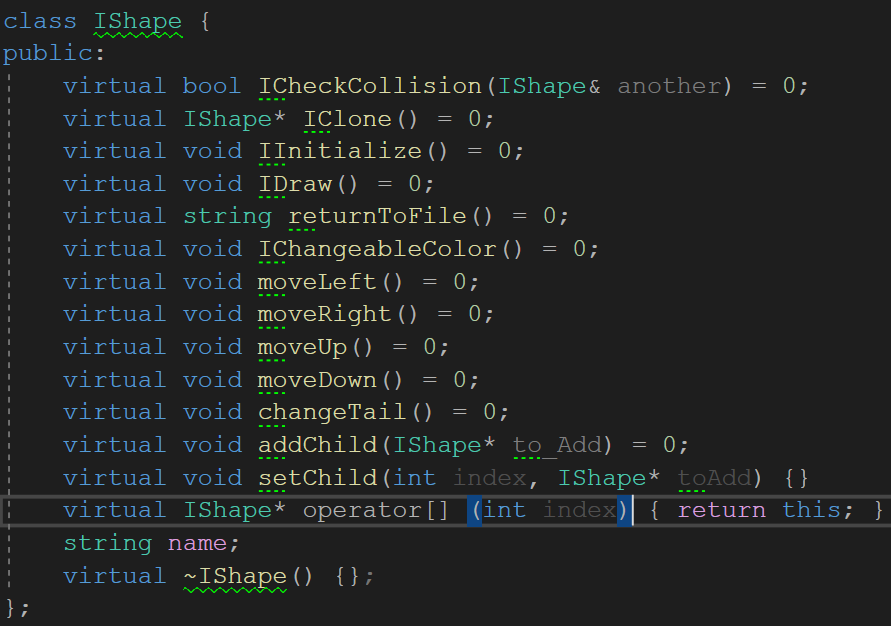


Рисунок 1 – клас IShape.

Від класу *IShape* успадковується клас *IPrimitive* (рис. 2).

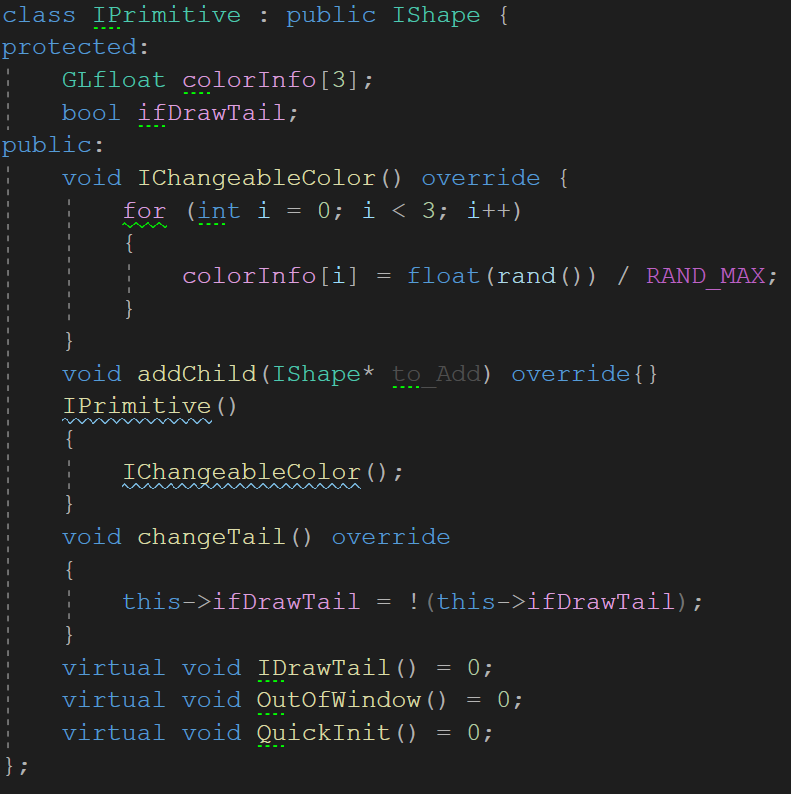


Рисунок 2 – клас IPrimitive

Також від базового класу *IShape* успадковується *Composite* який є контейнером для інших *IShape* (рис. 3).

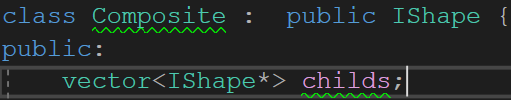


Рисунок 3 – Клас Composite

Зазначений вектор *childs* на рис. 3 тримає всі об’єкти агрегата.

Від *IPrimitive* успадковується 2 класа: *Rectangle* и *Circle*.

Це класи які відповідають за відрисовку *Circle* (рис. 4) та *Rectangle*(рис. 5) .



Рисунок 4 – Клас Circle



Рисунок 5 – Клас Rectangle

## Патерн Prototype

Для створення нових об'єктів патерн Prototype використовує прототипи.

Прототип – це вже існуючий в системі об'єкт, який підтримує операцію клонування, тобто вміє створювати копію самого себе. Таким чином, для створення об'єкта деякого класу досить виконати операцію *clone ()* відповідного прототипу.

Патерн Prototype реалізує подібну поведінку в такий спосіб: всі класи, об'єкти яких потрібно створювати, повинні бути підкласами одного загального абстрактного базового класу. Цей базовий клас повинен оголошувати інтерфейс методу *clone ().*

Переваги патерна Prototype:

* Для створення нових об'єктів клієнтові не обов'язково знати їх конкретні класи.
* Можливість гнучкого управління процесом створення нових об'єктів за рахунок можливості динамічних додавання і видалення прототипів до реєстру.

Патерн prototype використовується для створення глибоких деревовидних копій компонентів сцени. Завдяки цьому патерну, можна створити глибоку копію об'єкта, навіть якщо він зберігається в інтерфейсній посиланням, за рахунок перевизначеного методу *clone ().*

В базовому класі IShape наявний чистий віртуальний метод *IClone().*

*Virtual IShape\* IClone() = 0*;

В класах *Composite,Circle* та *Rectangle* наявний *IClone() override*.

При створені агрегата використовується копіювання (рис. 6).

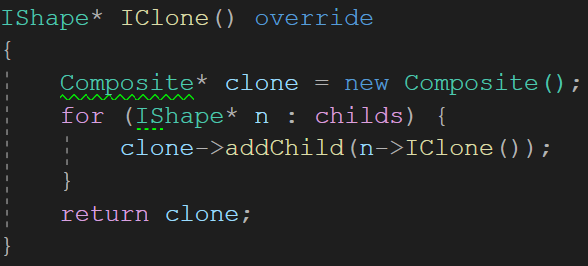


Рисунок 6 – Застосування методу IClone в створенні агрегата

## Патерн Singleton

Часто в системі можуть існувати суті тільки в єдиному екземплярі, наприклад, система ведення системного журналу повідомлень або драйвер дисплея. У таких випадках необхідно вміти створювати єдиний екземпляр деякого типу, надавати до нього доступ ззовні і забороняти створення декількох екземплярів того ж типу.

Архітектура патерна Singleton заснована на ідеї використання глобальної змінної, що має такі важливі властивості:

* Така змінна доступна завжди. Час життя глобальної змінної - від запуску програми до її завершення.
* Надає глобальний доступ, тобто, така змінна може бути доступна з будь-якої частини програми.

В даній роботі патерн Singleton використовується, як і описано вище, для того, щоб мати певну сутність тільки в одному екземплярі і мати до неї доступ з усієї програми.

Window, зазначений в класі *Scene* є глобальним об'єктом який створюється на початку програми та використовується протягом всього періоду використання програми (рис.7).

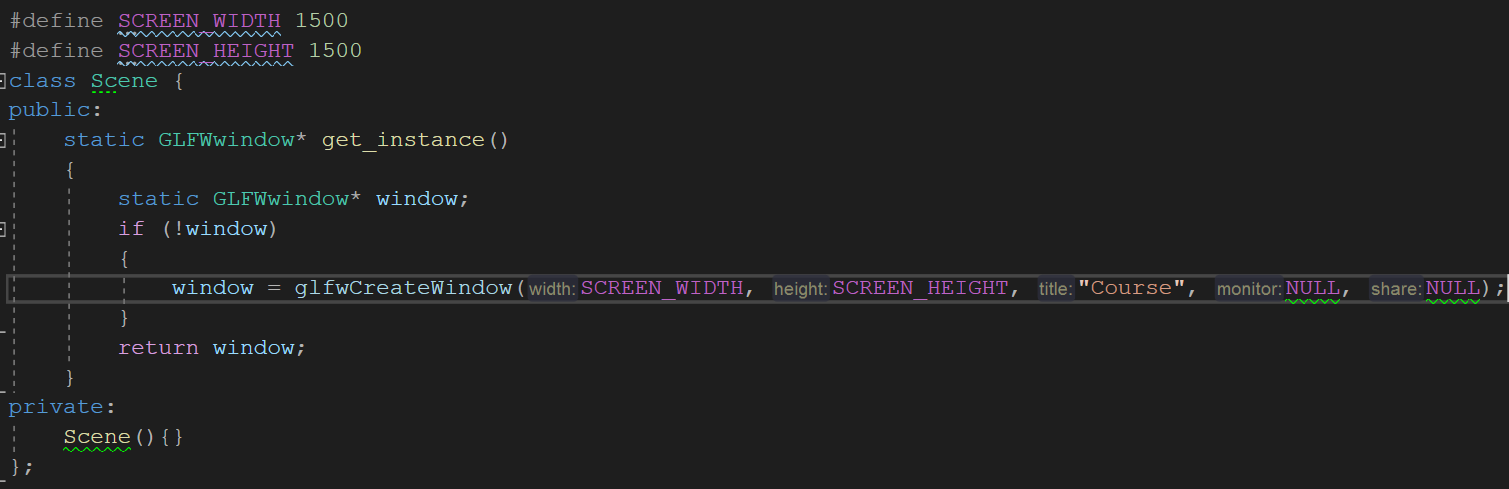


Рисунок 7 – Клас Scene який реалізує паттерн Singleton

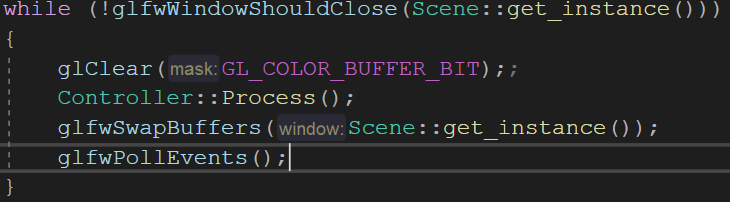
Головний цикл відрисовки використовує window з класу Scene (рис. 8).

Рисунок 8 – Головний цикл відрисовки

## Патерн Memento

Для збереження та завантаження *IShape* використовується .txt файл.

У базовому класі *IShape* присутній чистий віртуальний метод *returnToFile().*

*virtual string returnToFile() = 0*;

У класах *Circle* та *Rectangle* наявна override версія для повернення строки \*name\* \*координати\* в .txt файл. Зберігаємо лише координати?

saveState () – метод збереження координат у файл .txt (рис.9)

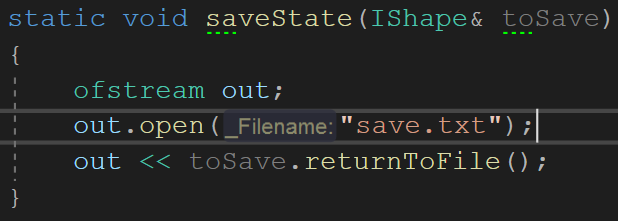


Рисунок 9 – Метод збереження в файл

*LоadStateHelper ()* – метод завантаження координат об’єкта з файла .txt (рис.10).

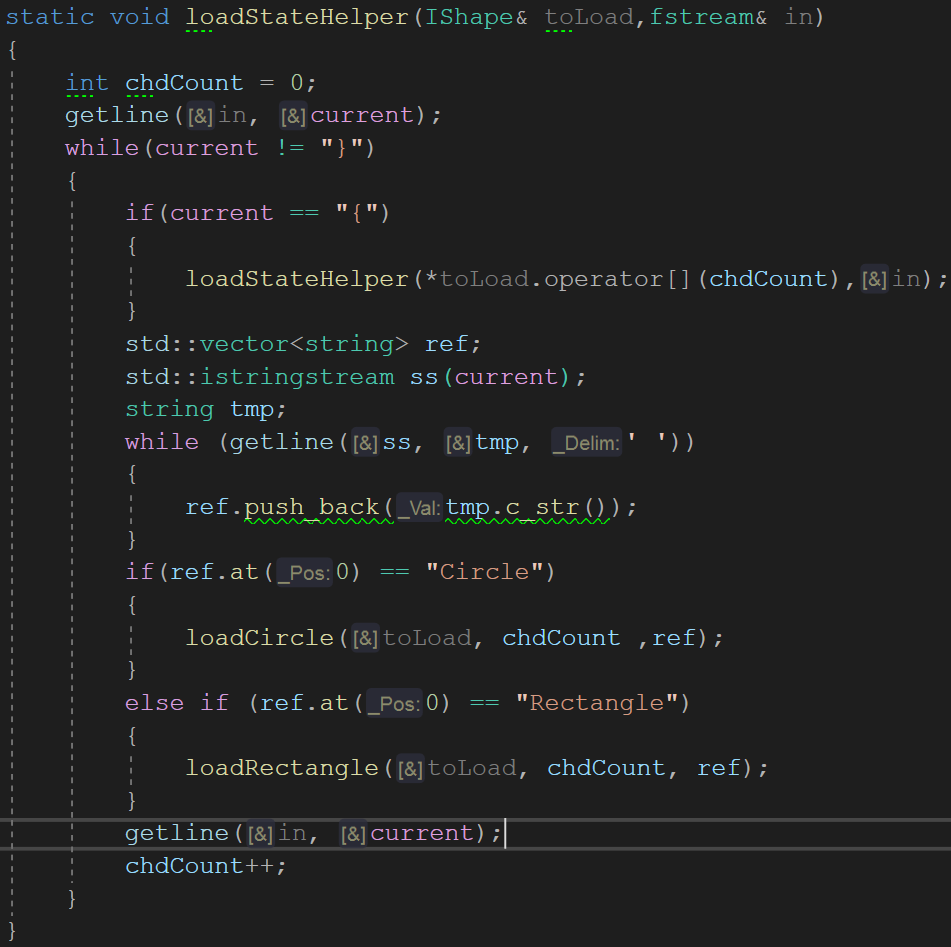


Рисунок 10 – Метод завантаження з файла

# Вихідний текст програми розв’язку задачі

*Scene.h :*

static GLFWwindow\* get\_instance() – повертає вікно для відрисовки, якщо воно не створено, створює його.

*IShape.h :*

IShape\* IClone() – повертає клон наявного об'єкта.

void IInitialize() – ініціалізація об'єкта.

void IDraw() – головний метод відрисовки наявного об'єкта.

string returnToFile() –повертає строку для збеореження в файл .txt.

void IChangeableColor() – міняє колір на рандомний.

virtual void sizeIncrease() = 0; - збільшує розімр об’єкта.

void moveLeft() – рух вліво.

void moveRight() – рух вправо.

void moveUp() – рух вгору.

void moveDown() – рух вниз.

void changeTail() – міняє рух зі слідом.

void addChild(IShape\* to\_Add) – додає child до Composite.

void setChild(int index, IShape\* toAdd) – міняє child.

IShape\* operator[] (int index) – перевантаженя оператора [].

string name – ім’я об’єкта.

bool ICheckCollision(IShape& another) – перевіряє колізію з іншим об’єктом.

*IPrimitive.h :*

void IDrawTail() – відрисовка сліду.

void OutOfWindow() – дія при виході за рамки вікна.

*Controller.h :*

void initControlled(IShape\* base) – ініціалізація контролера.

void changedControlled() – змінити активний об’єкт.

void addChildToControlled(IShape\* toAdd) – додати child до активного об’єкту.

void addShape(IShape\* toAdd) – додати Shape до сцени.

void Process() – головний метод відрисовки об’єктів сцени.

void Group() – згрупувати об’єкти сцени.

void displayMenu() – показ меню.

# Опис інтерфейсу

Меню для вибора дій з’являється на початку виконання програми (рисунок 9) перед появою сцени. У вас попередній рис.10, а тут посилаєтесь на 9

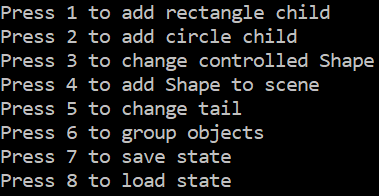


Рисунок 9 – Головне меню

Керування забезпечується наступними кнопками, відповідно до меню на рис. 9:

1. додавання прямокутника до агрегата;
2. додавання круга до агрегата;
3. зміна контролюємого об’єкта;
4. додати прямокутник до сцени;
5. включити/виключити переміщення зі слідом;
6. згрупувати всі наявні об’кти на сцені;
7. зберегти координати контролюємого об’єкту;
8. завантажити координати з файла до активного об’єкту.

# Опис тестових прикладів

Додамо Circle та Rectangle до активного агрегата (рис. 10).

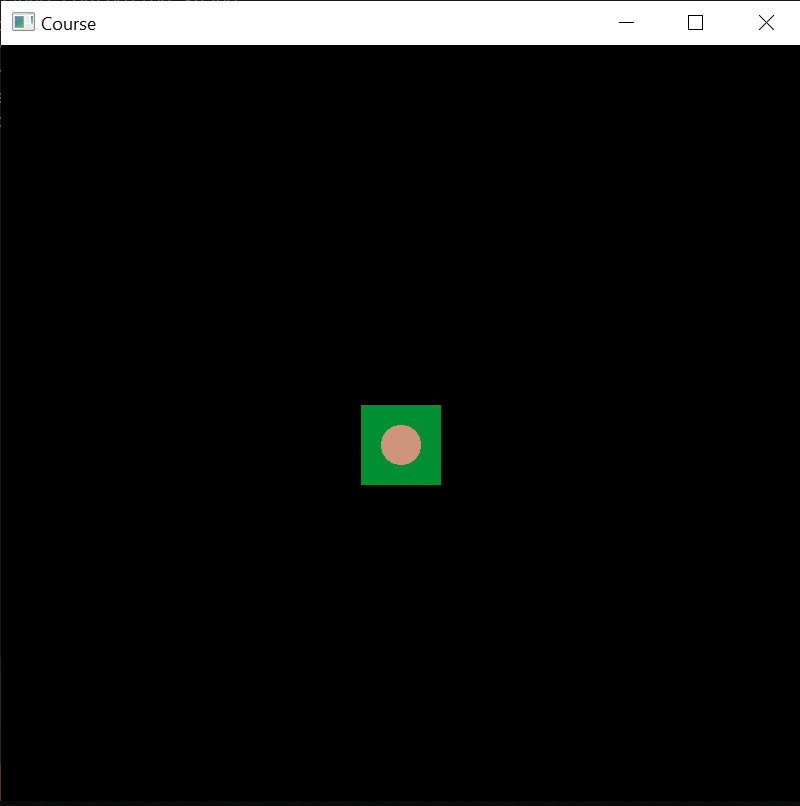


Рисунок 10 – Додали Rectangle та Circle.

Додамо об’єкт до сцени (рисунок 12).

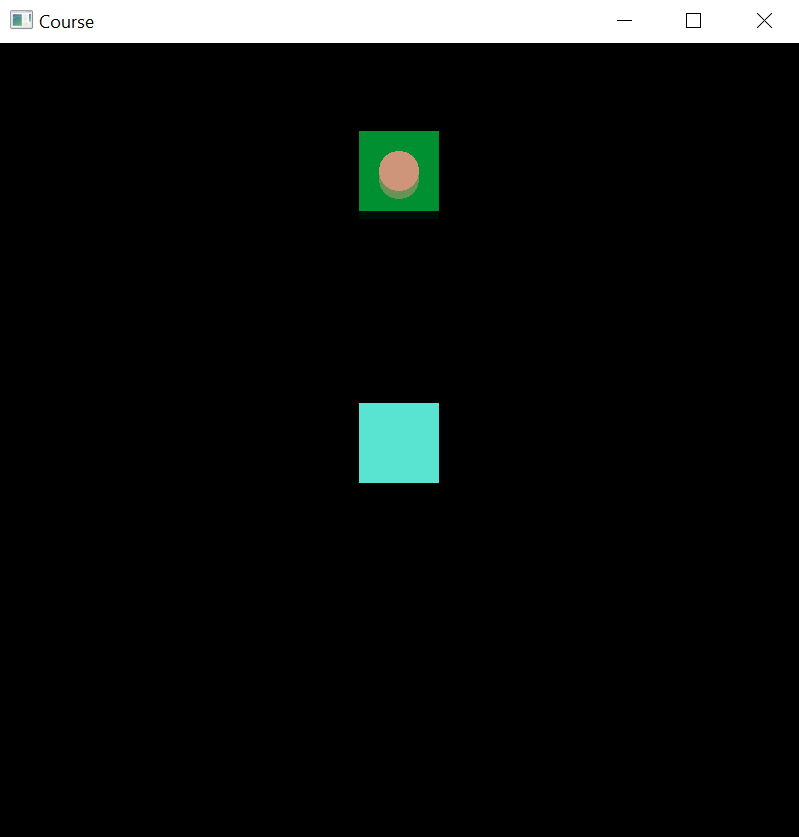


Рисунок 12 – Додали Rectangle до сцени (не в агрегат)

Змінемо положення активного об’єкта на сцені (рис. 13­).

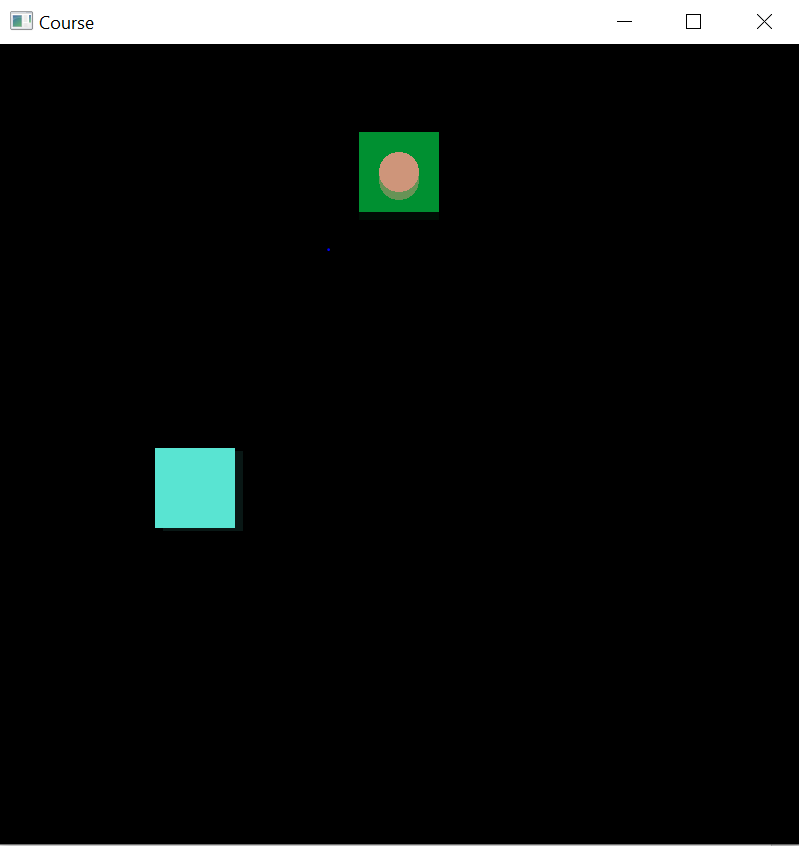


Рисунок 13 – Змінили координати доданого до сцени Rectangle

Згрупуємо об’єкти сцени (рис. 14).

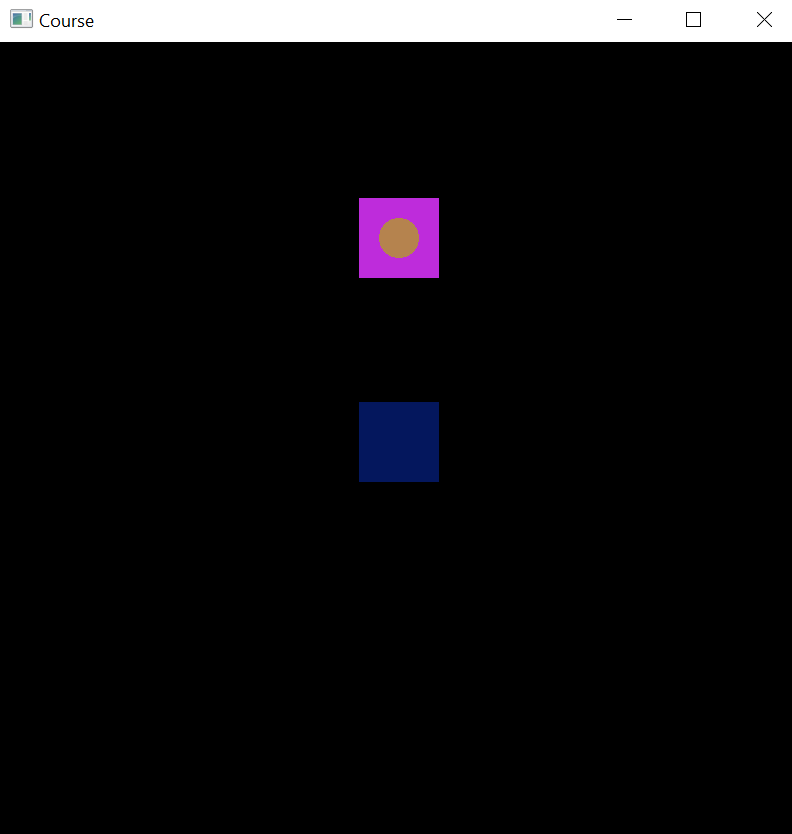


Рисунок 14 – При згрупуванні об’єктів міняється їх колір.

Збереження стану об’єктів (рис.15) в файл \*.txt.

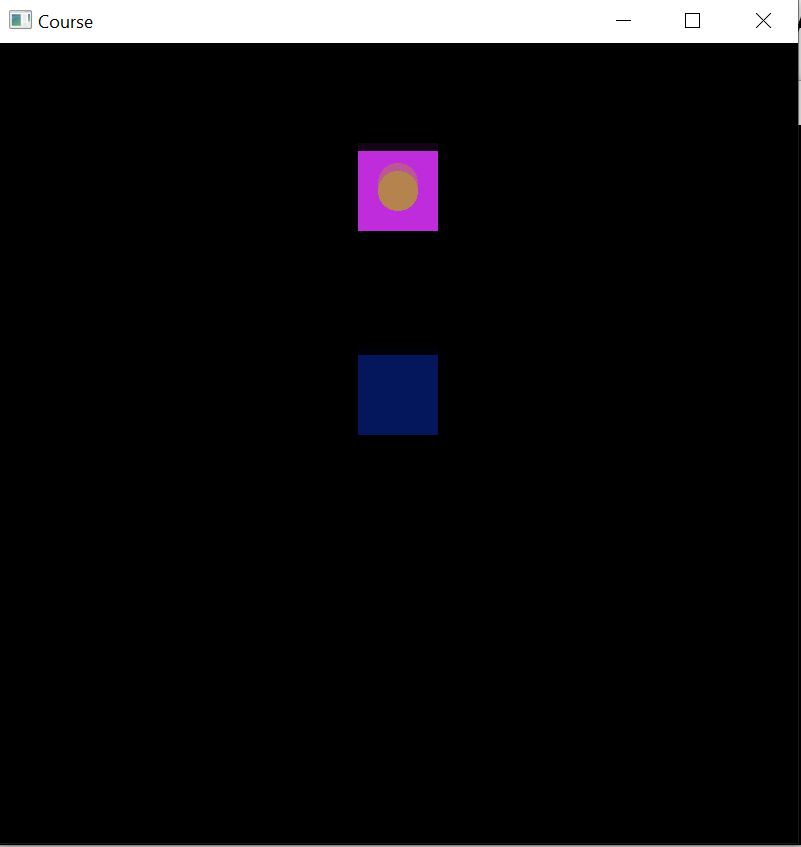


Рисунок 15 – Зберегли ці координати в файл

А чому ми зберігаємо лише координати? А стан об’єкта, його колір? Розмір?

На рисунку 16 показано в якому вигляді зберігається стан об’єкта в файл \*.txt.

Фігурні скобки зазначають об’єкт Composite.

Рядок файлу містить інформацію у вигляді: \*назва об’єкту\* \*збережені координати\*.

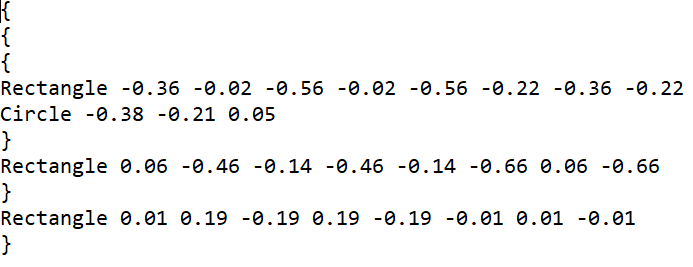


Рисунок 16 – Збережені координат

На рисунку 17 зображено стан сцени збереження стану об’єкта на рисунку 15 ми змінюємо його координати.

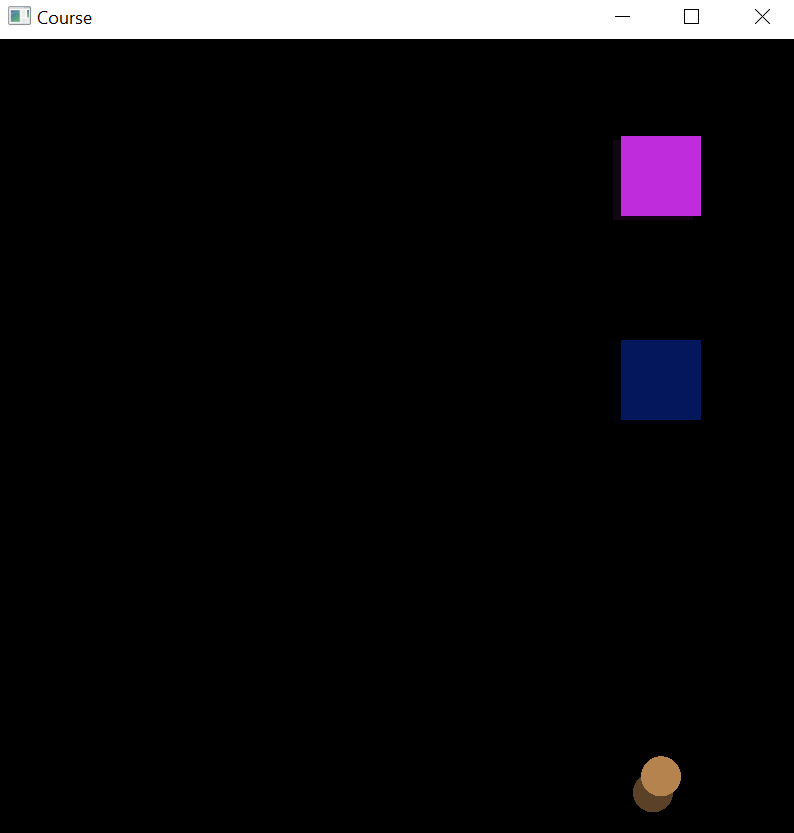


Рисунок 17 – Результат зміни координат

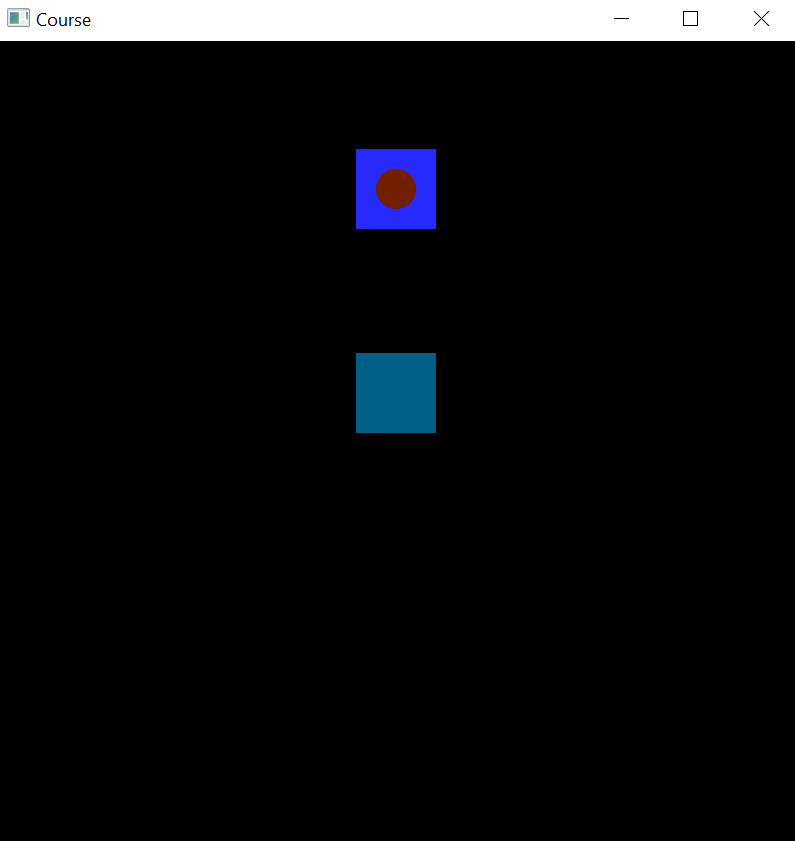


Рисунок 18 – Відновили координати з файлу

На рис. 18 зображено відновлений стан об'єкта з файла.

# Аналіз помилок

Написати про реалізацію збереження в файл не лише координат, а й інших характеристик об’єктів.