**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-21-1 Спеціальності 121 Інженерія програмного забезпечення

Товпаша Є. Ю.

(прізвище та ініціали)

Керівник

\_

\_

Кількість балів

Національна шкала

Члени комісії :

(підпис) (прізвище та ініціали)

(підпис) (прізвище та ініціали)

(підпис) (прізвище та ініціали)

м. Дніпро, 2023 р.

Зміст:

1. Постановка задачі 2
2. Теоретична частина 3
3. Практична реалізація 7
4. Тестування 13
5. Список літератури. 20

# ПОСТАНОВКА ЗАДАЧІ

Профести рефакторинг лабораторної роботи 4 (геометричні фігури) 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 створення фігур та агрегатів використовувався патерн Прототип (Prototype), передбачити реєстрацію доступних прототипів у сховищі;
  3. патерн Одинак (Singleton) забезпечував існування лише одного контролера сцени;
  4. патерн Знімок (Memento) використовувався для зберігання стану наявних фігур на сцені у файл на диску та відновлення сцени з файлу;
  5. додавання інших патернів є додатковою перевагою.

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.

# ТЕОРЕТИЧНА ЧАСТИНА

Проведено рефакторинг коду лабораторної роботи № 4 минулого семестру і використано наступні патерни проєктування: Композит, Прототип, Одинак і Знімок. Розглянемо будову класів, завдяки яким було реалізовано даний задум. Почнемо з основного класу – Figure (дивіться рисунок 2.1).

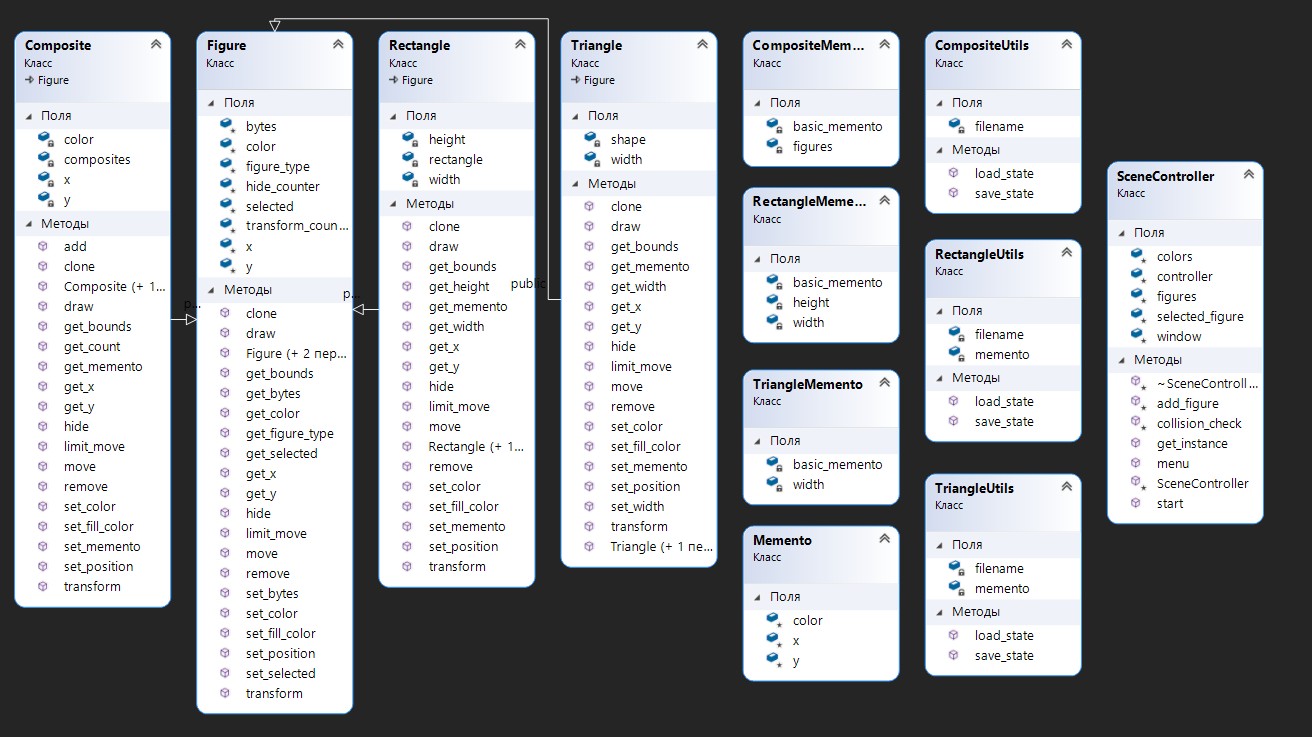


Рис. 2.1 – Діаграма класів

Згідно діаграми класів, клас Figure є спадковим для декількох класів, які реалізовують конкретні фігури. У цьому класі міститься перелік основних характеристик будь-якої фігури та віртуальні методи без реалізації, які треба перевизначати в усіх класах-спадкоємцях. Перелічимо ці методи:

* virtual void move – відповідає за пересування фігури по сцені за заданим законом, в якості параметрів приймає розмір вікна програми, коефіцієнт руху та швидкість руху;
* virtual void draw – відповідає за відображення фігури на сцені, в якості параметрів приймає посилання на вікно;
* Figure() – конструктор за замовчуванням, також присутні 2

конструктори з параметрами, один з них приймає тільки тип фігури, а другий – координати, тип фігури та її колір;

* virtual FloatRect get\_bounds – відповідає за отримання даних про межі фігури на сцені;
* Color get\_color – геттер для поля кольору;
* int get \_bytes – геттер для кількості байтів, яку займає фігура в файлі;
* string get\_figure\_type – геттер для типу фігури;
* bool get\_selected – геттер для поля, яке необхідне для регулювання процесу обирання фігур;
* virtual int get\_x, virtual int get\_y – відповідають за отримання координат фігури;
* virtual void hide – відповідає за приховування фігури на сцені;
* virtual void limit\_move – відповідає за обмеження руху фігури, контролює щоб вона не виходила за межі наявної сцени, в якості параметра приймає розмір вікна;
* virtual void remove – відповідає за видалення фігури зі сцени, в якості параметра приймає покажчик на вікно;
* virtual void set\_color, set\_fill\_color – перший метод встановлює колір для фігури і запам’ятовує його, другий – лише встановлює, в якості параметра приймається відповідно колір, який необхідно встановити;
* virtual void set \_bytes – сеттер для поля з кількістю байтів, які займає фігура у файлі, приймає відповідно нову кількість байтів;
* virtual void set\_position – відповідає за розміщення фігури на сцені за конкретними координатами, в якості параметрів приймає ці координати;
* virtual void set\_selected – сеттер для поля, яке є допоміжним під час роботи програми;
* virtual void transform – відповідає за деформацію фігури;
* virtual Figure\* clone – відповідає за клонування фігури.

Клас Triangle є похідним від класу Figure і реалізує всі віртуальні методи з нього у відповідності до функціоналу типу CircleShape. Цей клас містить приватне поле з фігурою типу CircleShape і також поле з довжиною сторони трикутника.

Наступним розглянемо клас Rectangle. Він так само як і клас Triangle є похідним від класу Figure і реалізує всі віртуальні методи з нього у відповідності до функціоналу типу RectangleShape. Цей клас містить приватне поле з фігурою типу RectangleShape і також поля з довжиною та висотою прямокутника.

Клас Composite реалізує патерн проєктування Композит та є похідним від класу Figure і реалізує всі віртуальні методи з нього. Має захищене поле типу vector з елементами Figure, а також координати композиту на сцені. Завдяки такій реалізації ми можемо робити агрегати як з простих фігур, так і з інших агрегатів.

В усіх класах, які реалізують окремо взяті фігури чи агрегат додатково реалізовано методи get\_memento і set\_memento, які відповідають за коректну роботу патерну Знімок.

Для реалізації патерну Знімок було створено клас Memento, він зберігає основні характеристики, які є у кожної фігури. На додачу було створено три додаткові класи: RectangleMemento, TriangleMemento і CompositeMemento задля зберігання тих характеристик, які присутні тільки у окремо взятої фігури. Процес зберігання стану фігур контролюється у класах RectangleUtils, TriangleUtils та CompositeUtils, які мають методи load\_state (зберігає стан до файлу) та save\_state (зчитує стан з файлу і зберігає його у відповідній фігурі). І наостанок розглянемо клас SceneController. Він реалізований згідно патерну проєктування Одинак, що забезпечує існування лише 1 контроллера сцени. У якості полів маємо статичний покажчик на цей же клас, покажчик на вікно, вектор з фігурами, якими керує користувач, масив кольорів, змінна, що

містить індекс обраної фігури. (дивіться рисунок 2.2).

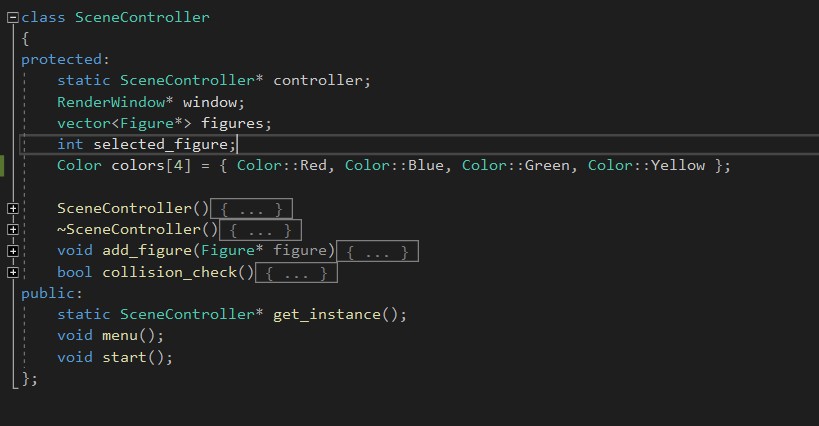


Рис. 2.2 – Клас SceneController В ньому реалізовано наступні методи:

* SceneController(), ~SceneController() – конструктор і деструктор, мають область видимості protected в межах реалізації патерну Одинак;
* void add\_figure – відповідає за додавання нової фігури до списку, в якості параметра приймає нову фігуру;
* bool collision\_check – відповідає за перевірку наявності зіткнень між фігурами;
* static SceneController\* get\_instance() – відповідає за отримання покажчика на єдиний контроллер сцени (патерн Одинак);
* void menu – відповідає за відображення меню для користувача з вказівками щодо того, за що яка клавіша відповідає;
* void start – основний метод, який відповідає за роботу сцени.

# ПРАКТИЧНА РЕАЛІЗАЦІЯ

Нижче показано, яким чином було реалізовано всі 4 патерни: Композит, Прототип, Одинак та Знімок.

Почну з патерну Прототип. Він дозволяє нам отримати новий об’єкт на основі даних існуючого. Для його реалізації у клас Figure було додано віртуальний метод clone (дивіться рисунок 3.1).

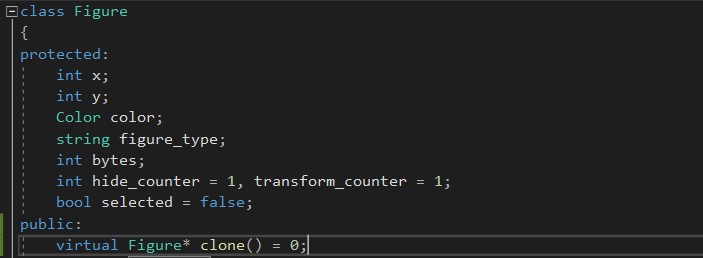


Рис. 3.1 – Сlass Figure

Метод clone згодом перевизначається в усіх класах-нащадках (дивіться рисунки. 3.2-3.4). Це дозволяє реалізовувати процедуру клонування для кожної фігури окремо, в залежності від її полів і особливостей.

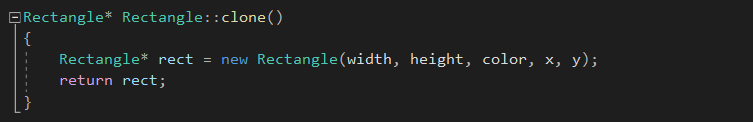


Рис. 3.2 - Метод clone в класі Rectangle

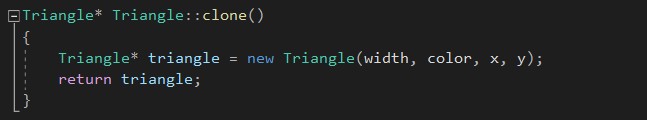


Рис. 3.3 – Метод clone в класі Triangle



Рис. 3.4 – Метод clone в класі Composite

Далі йде патерн Композит. Даний патерн надає можливість працювати з групою різних об’єктів як з одним суцільним об’єктом. В моїй програмі клас Composite є похідним від класу Figure. При створенні композиту нові фігури додаються до вектору figures (дивіться рисунок 3.5).

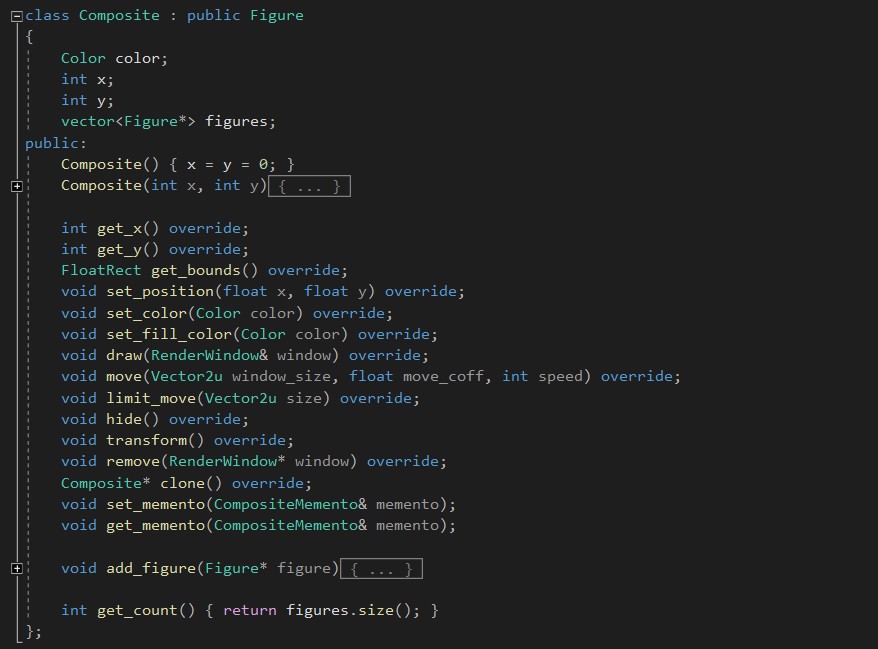


Рис. 3.5 – Сlass Composite

Дії з усіма фігурами, що розташовані у векторі figures, відбуваються у перевизначених методах з батьківського класу шляхом перебору усіх елементів, що входять до складу композиту і виконання відповідних дій з кожним з них. Нижче наведені кілька з цих методів (дивіться рисунок 3.6).

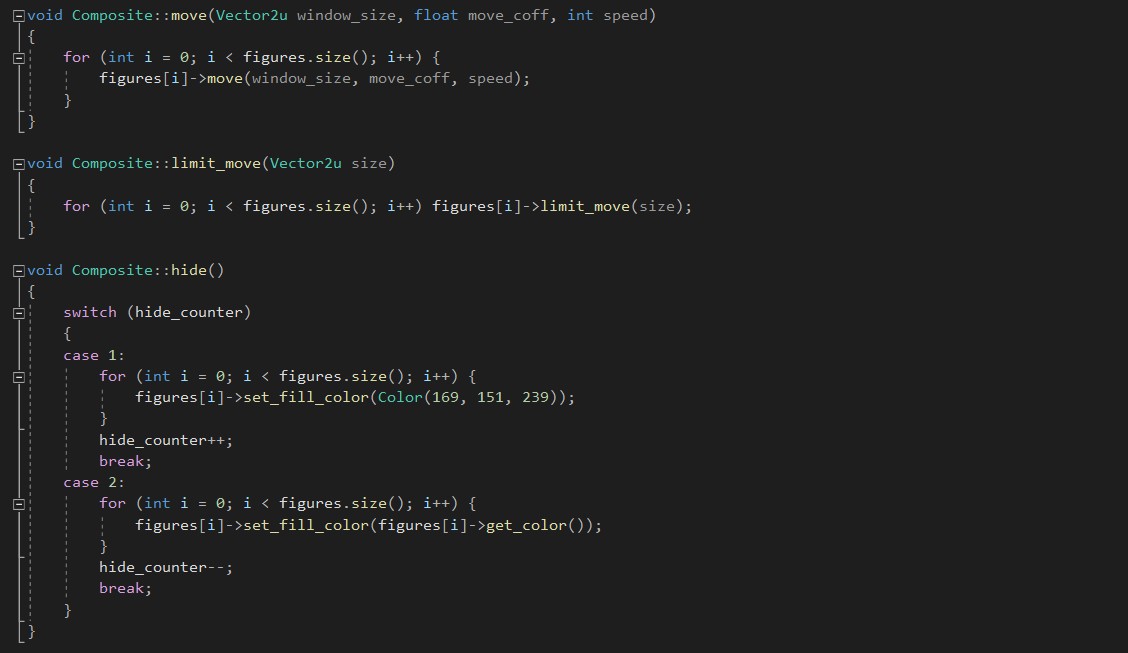


Рис. 3.6 – методи класу Composite

Патерн Знімок реалізовує збереження та відновлення стану сцени у файл та з файлу відповідно. Задля його реалізації було створено клас Memento (дивіться рисунок 3.7), який містить інформацію, яку мають всі фігури та три класи: RectangleMemento (дивіться рисунок 3.8), TriangleMemento, CompositeMemento, які містять в собі окрім загальної інформації для всіх фігур ще й інформацію, яку може мати тільки ця окремо взята фігура. Для контролю збереження у файл і зчитування з нього було створено класи RectangleUtils (дивіться рисунок 3.9), TriangleUtils, CompositeUtils. Також до класів окремо взятих фігур було додано методи get\_memento та set\_memento (дивіться рисунок 3.10), які роблять знімок стану з фігури та зберігають стан до фігури відповідно.

Реалізацію даного патерну буде продемонстровано на прикладі прямокутника.

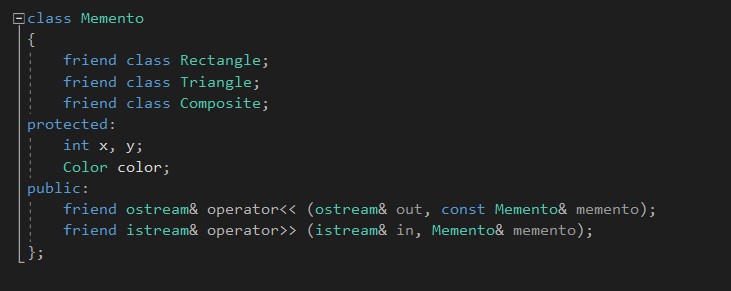


Рис. 3.7 – Сlass Memento

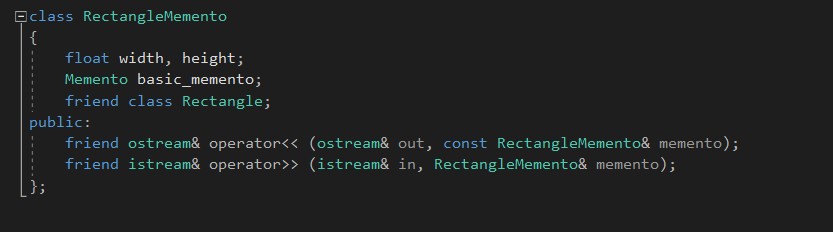


Рис. 3.8 – Сlass RectangleMemento

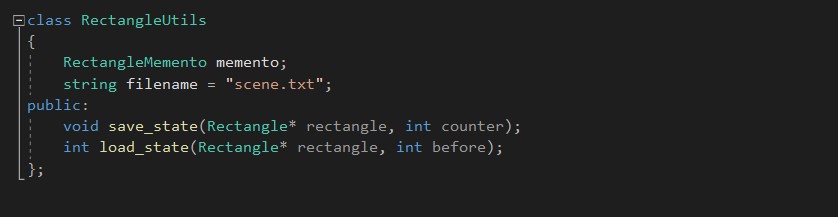


Рис. 3.9 – Сlass RectangleUtils

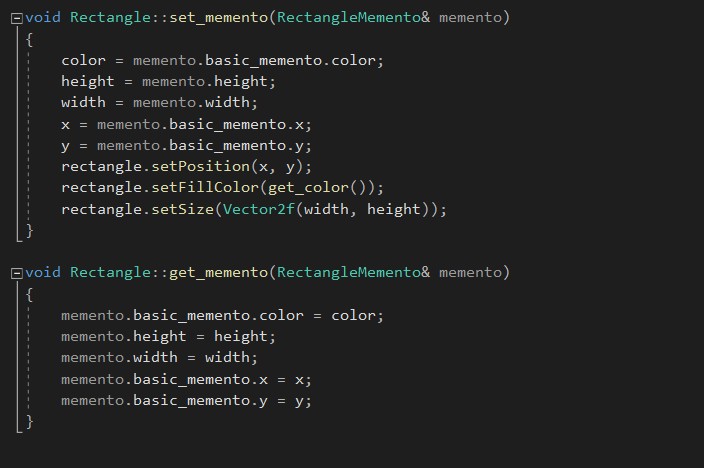


Рис. 3.10 – Методи set\_memento і get\_memento в класі Rectangle Патерн Одинак стежить за тим, щоб у нас була можливість створювати

тільки один контроллер сцени. Він надає до нього функцію доступу через відповідний метод. Для реалізації даного патерну було реалізовано protected конструктор, статичне поле з покажчиком на сцену та статичний метод get\_instance, який повертає цей покажчик (дивіться рисунки 3.11 – 3.14).

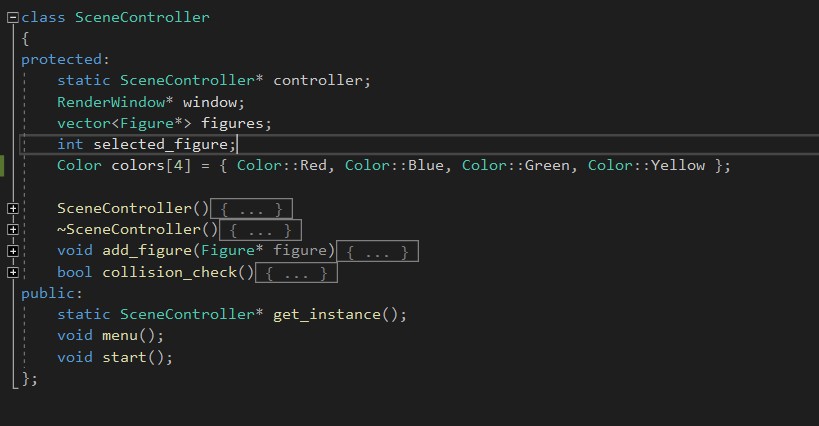


Рис. 3.11 – Сlass SceneController

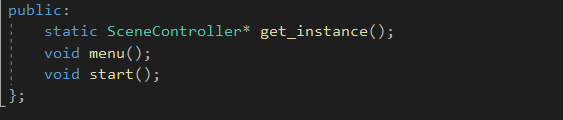


Рис. 3.12 – Оголошення методу get\_instance

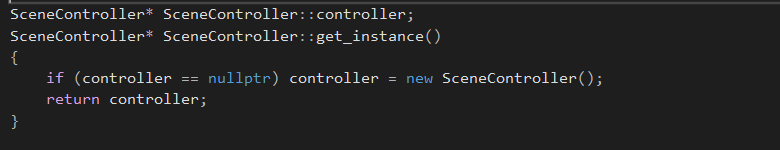


Рис. 3.13 - Реалізація методу get\_instance

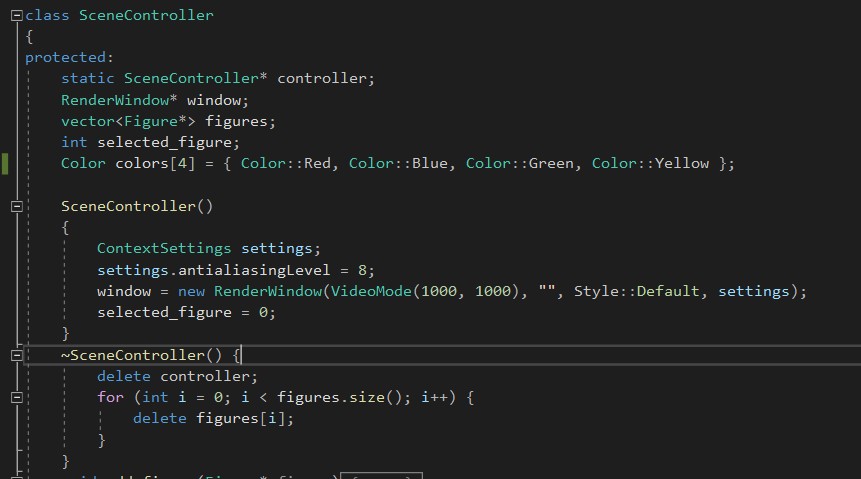


Рис. 3.14 – Рrotected конструктор і деструктор класу SceneController

# ТЕСТУВАННЯ

Користувач має можливість керувати перебігом програми за допомогою клавіатури. Реалізовано метод, який відповідає за донесення інструкцій користувачеві.

Метод, що реалізує меню, наведено на рисунку 4.1.

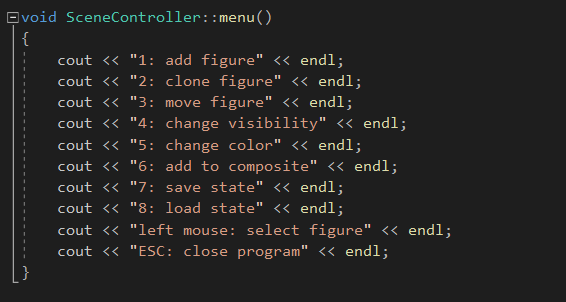


Рис. 4.1 – Метод menu

На рисунках 4.2 - 4.3 показаний процес додавання фігури

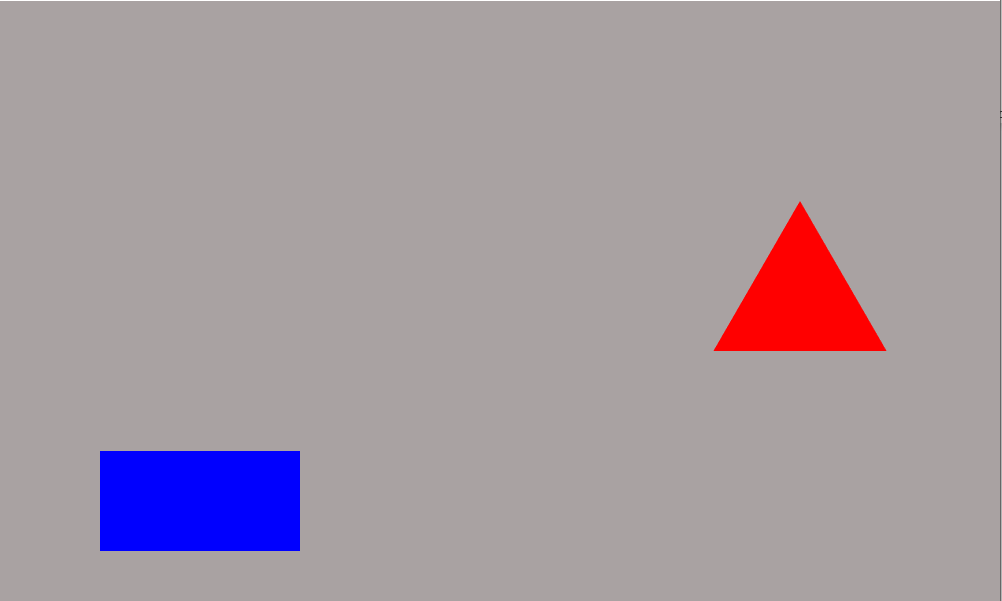


Рис. 4.2 – Початковий стан

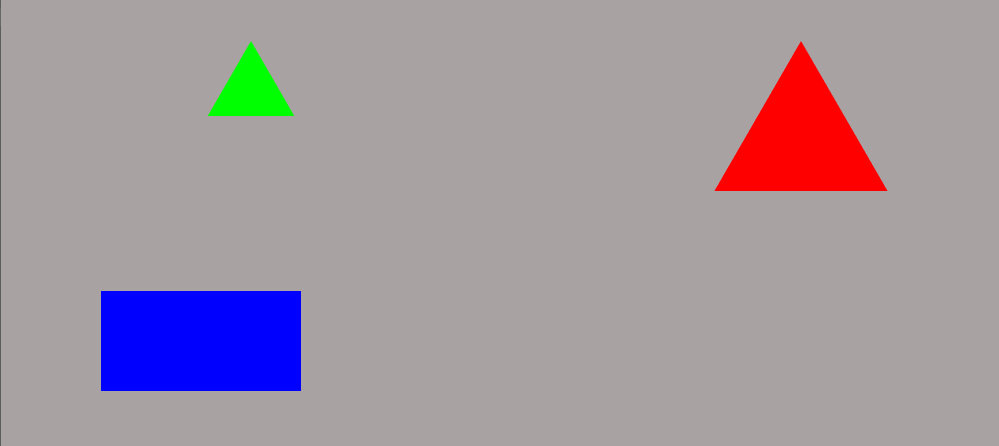


Рис. 4.3 – Кінцевий стан

На рисунках 4.4 – 4.5 показано процес додавання фігур до композиту

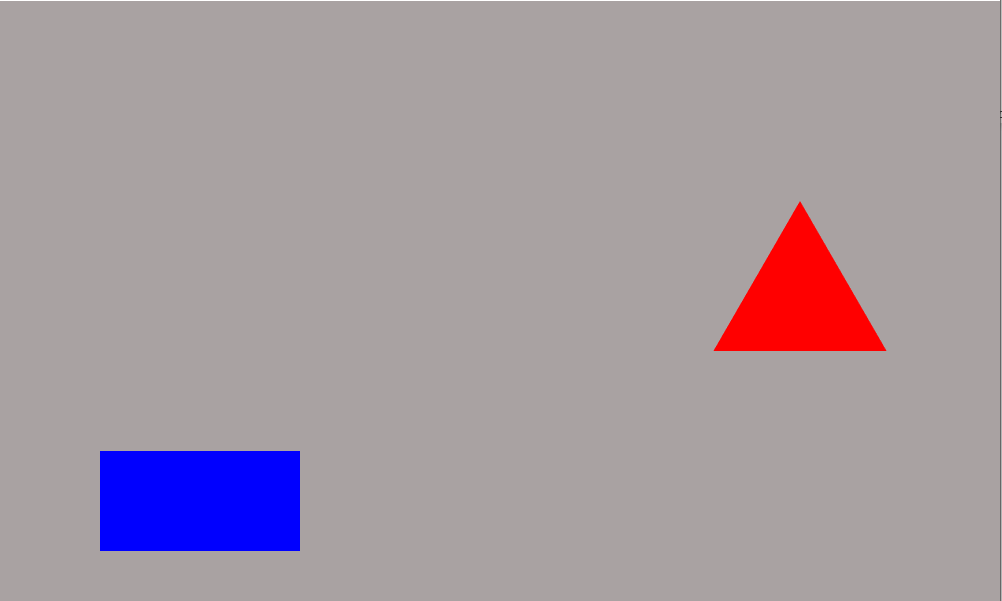


Рис. 4.4 – Початковий стан

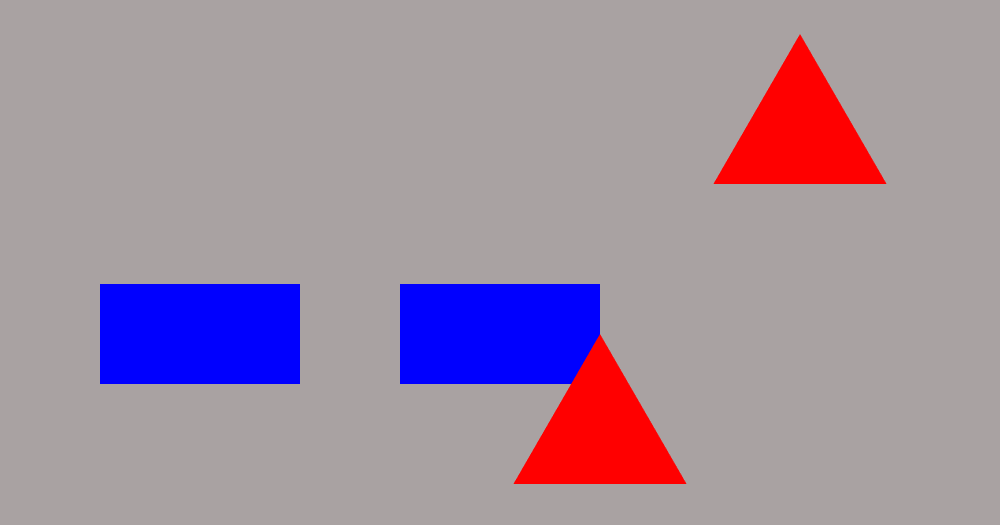


Рис 4.5 – Кінцевий стан На рисунках 4.6 - 4.7 показаний рух фігури

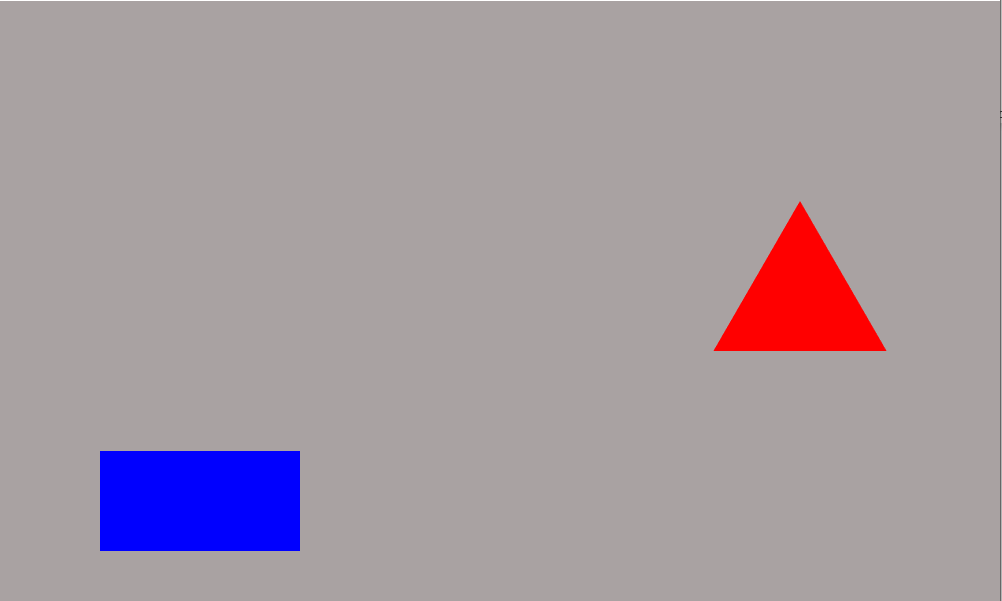


Рис. 4.6 – Початковий стан

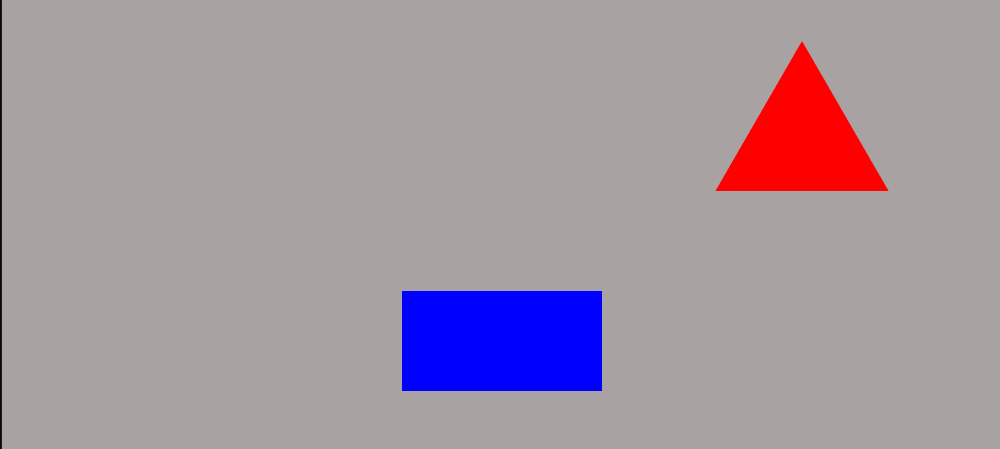


Рис. 4.7 – Кінцевий стан

На рисунках 4.8 – 4.9 показана зміна кольору фігури за командою

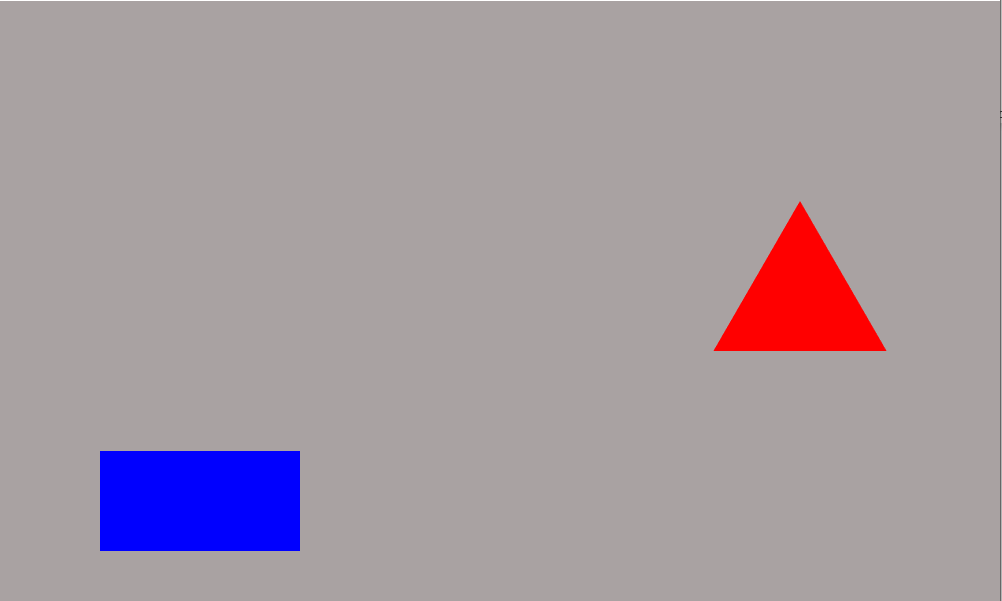


Рис. 4.8 – Початковий стан



Рис. 4.9 – Кінцевий стан

На рисунках 4.10 – 4.11 показана деформація фігури після зіткнення з іншою під час руху

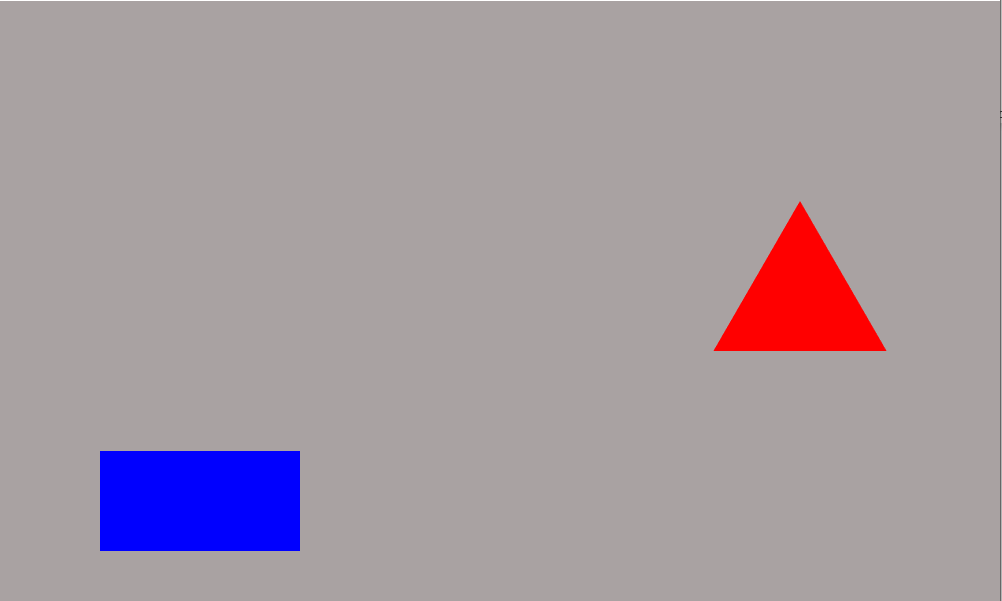


Рис. 4.10 – Початковий стан

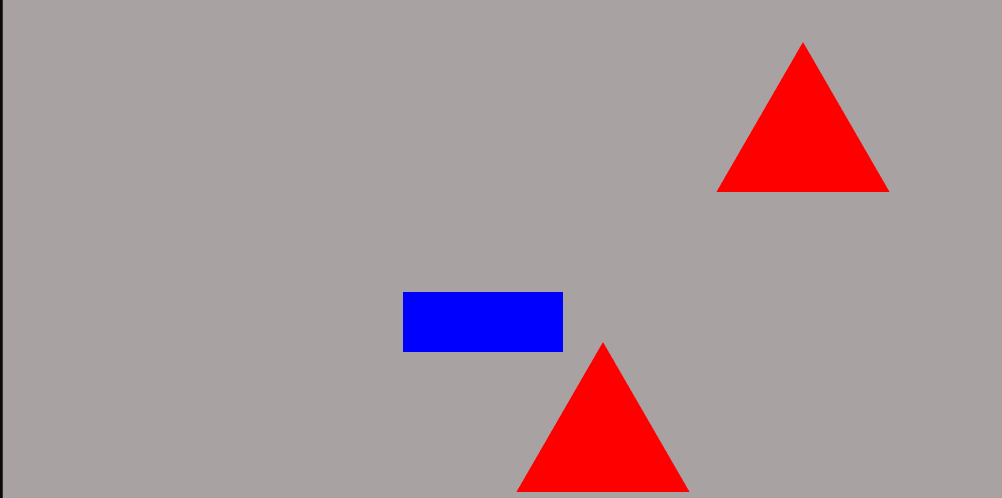


Рис. 4.11 – Кінцевий стан

На рисунках 4.12 – 4.14 показане збереження стану сцени в файл, після цього відбуваються певні зміни і потім сцена завантажується з файлу

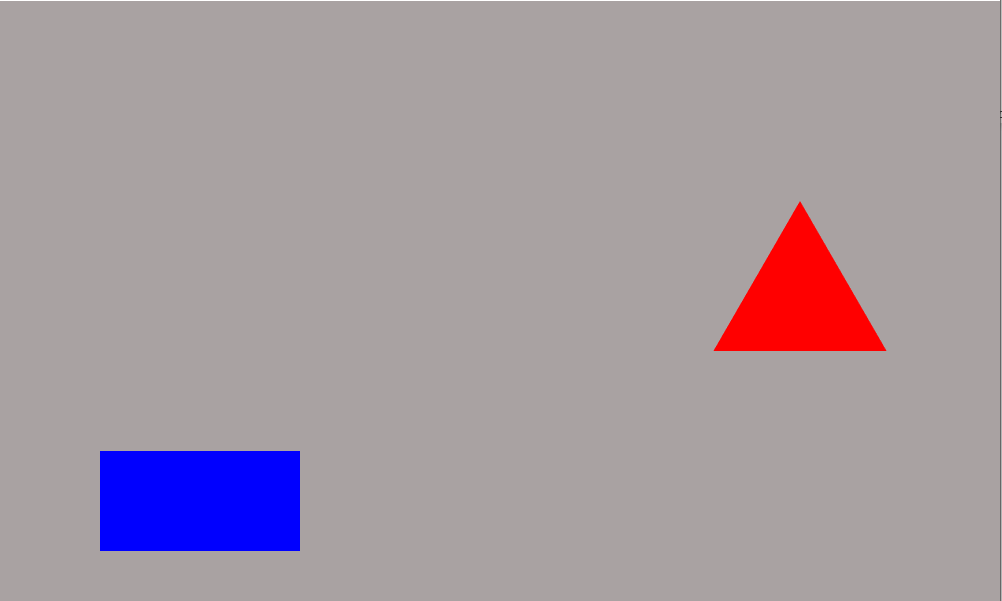


Рис. 4.12 – Початковий стан, відбувається збереження стану сцени

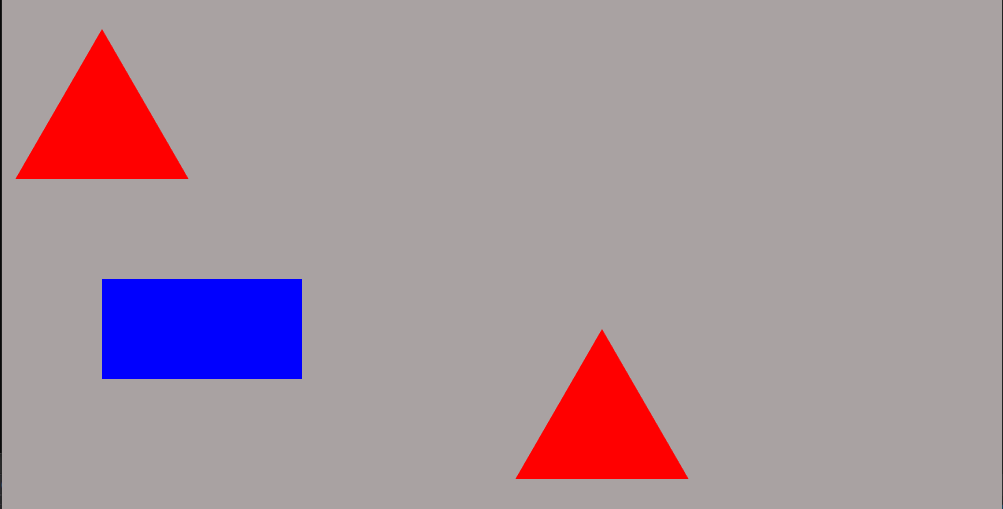


Рис. 4.13 – Проміжний стан після певних дій

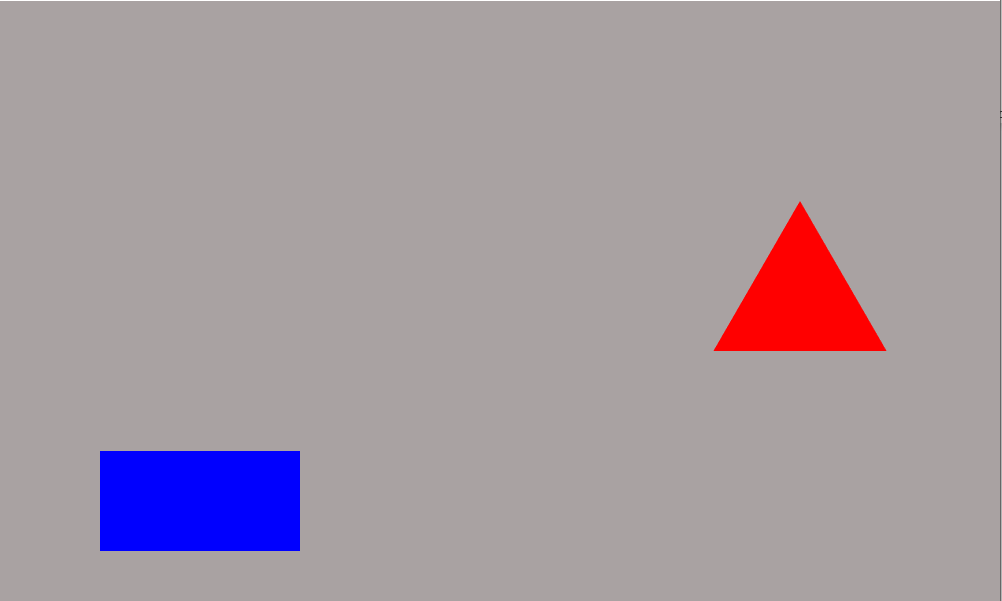


Рис. 4.14 – Кінцевий стан після відновлення сцени з файлу

# СПИСОК ЛІТЕРАТУРИ

1. Патерн Прототип. URL: [https://refactoring.guru/ru/design-](https://refactoring.guru/ru/design-patterns/prototype) [patterns/prototyp](https://refactoring.guru/ru/design-patterns/prototype)e (дата звернення: 26.02.2023).
2. Патерн Одинак. URL: <https://refactoring.guru/ru/design-patterns/singleton> (дата звернення: 26.02.2023).
3. Патерн Знімок. URL: <https://refactoring.guru/ru/design-patterns/memento> (дата звернення: 26.02.2023).
4. Патерн Композит. URL: [https://refactoring.guru/ru/design-](https://refactoring.guru/ru/design-patterns/composite) [patterns/composite](https://refactoring.guru/ru/design-patterns/composite) (дата звернення: 26.02.2023).