ЛАБОРАТОРНА РОБОТА № 1

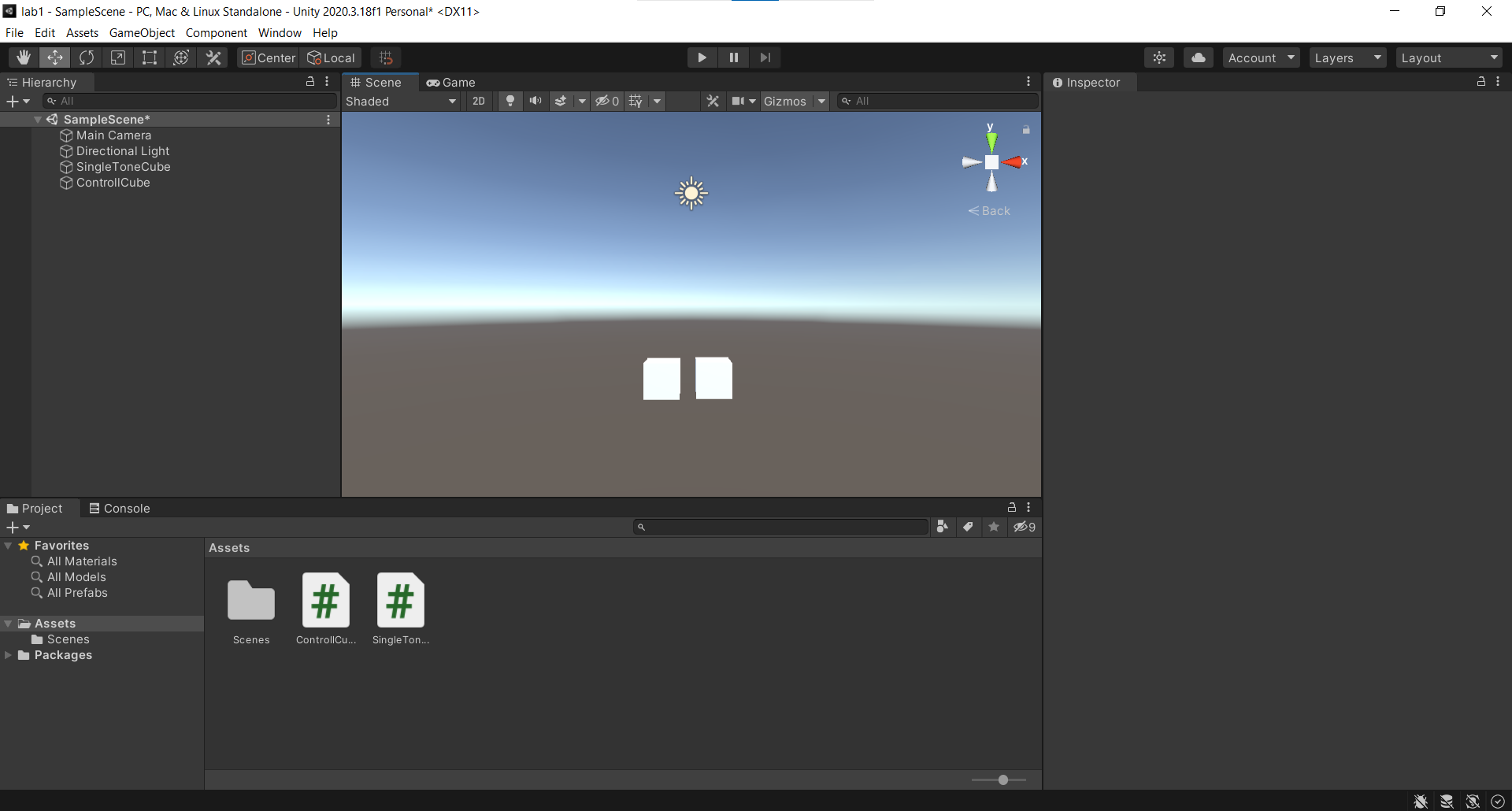
**Тема: «**Конструювання програмного забезпечення**»**

**Мета роботи:** ознайомитися із ігровим рушієм Unity, налаштувати середовище для роботи та отримати практичні навички зі створення об’єктів та скриптів, а також ознайомитися з практичним використання шаблону програмування.

**Хід роботи:**

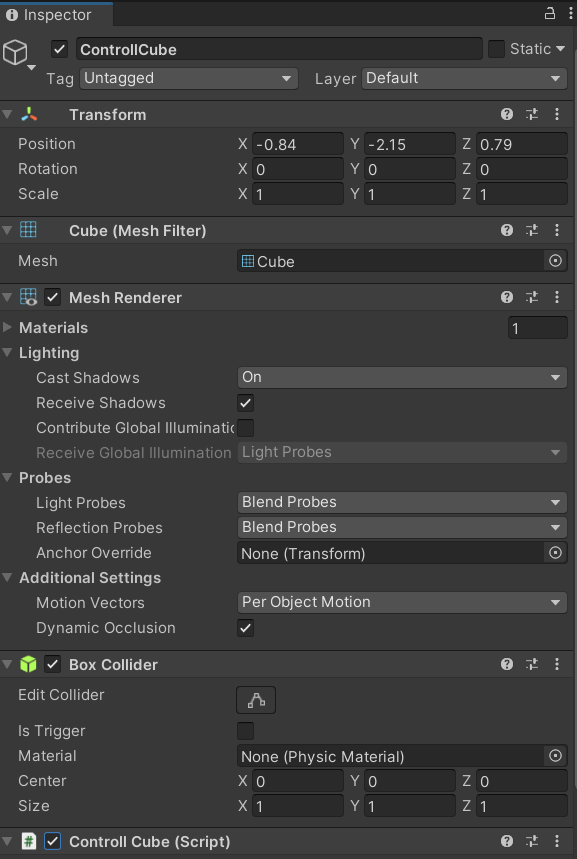
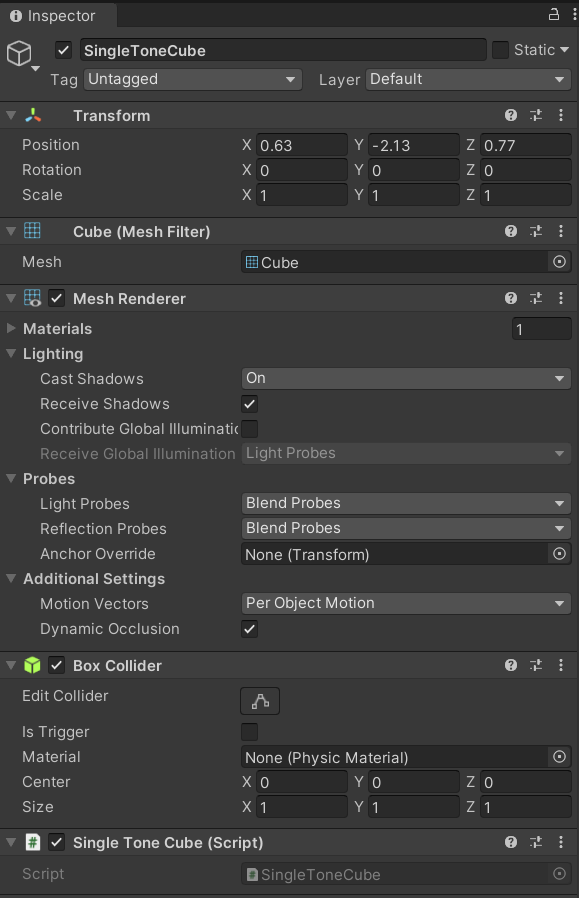
***Завдання 2:*** Створити новий проект у 3D режимі. Створити тестові об’єкти та скрипти, якими вони будуть керуватися.

Створений проект у 3D режимі:



***Завдання 3:*** Додати скрипти до відповідних об’єктів. Для цього оберіть, наприклад, «ControllCube» в меню ієрархії, або на сцені. Справа відобразиться налаштування цього об’єкту та його компоненти: оберіть наступні опції, натиснувши Add Component: Scripts->ControllCube. Таким чином вы додали новий компонент до об’єкту. Повторіть цю операцію з другим кубиком.

Додані скрипти до двох тестових кубів:

****** ******

***Завдання 4:*** Відредагуйте вихідний код скриптів, відкривши їх у редакторі коду подвійним натисканням на файл.

Лістинг класу ControllCube:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ControllCube : MonoBehaviour

{

private MeshRenderer \_renderer;

void Start()

{

\_renderer = GetComponent<MeshRenderer>();

}

// Update is called once per frame

void Update()

{

\_renderer.material.color = Input.GetButton("Fire1") ? Color.blue : Color.white;

if (Input.GetButtonDown("Fire1"))

{

SingleToneCube.Instance.SetColorState(true);

}

else if (Input.GetButtonUp("Fire1"))

{

SingleToneCube.Instance.SetColorState(false);

}

}

}

Лістинг класу SingleToneCube:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class SingleToneCube : MonoBehaviour

{

public static SingleToneCube Instance { get; private set; }

private MeshRenderer \_renderer;

void Start()

{

\_renderer = GetComponent<MeshRenderer>();

SetColorState(false);

Instance = this;

}

public void SetColorState(bool state)

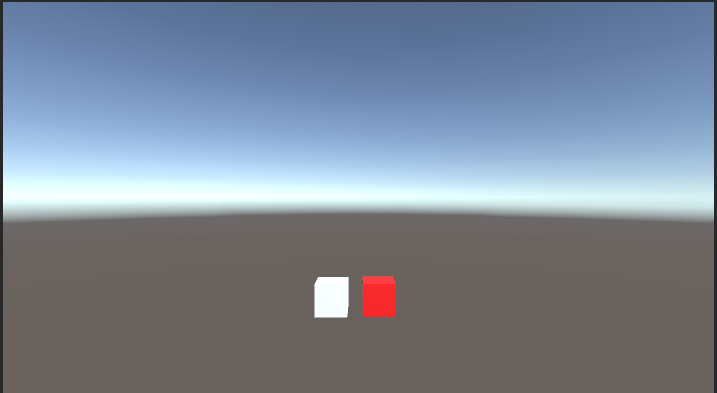
{

\_renderer.material.color = (state) ? Color.green : Color.red;

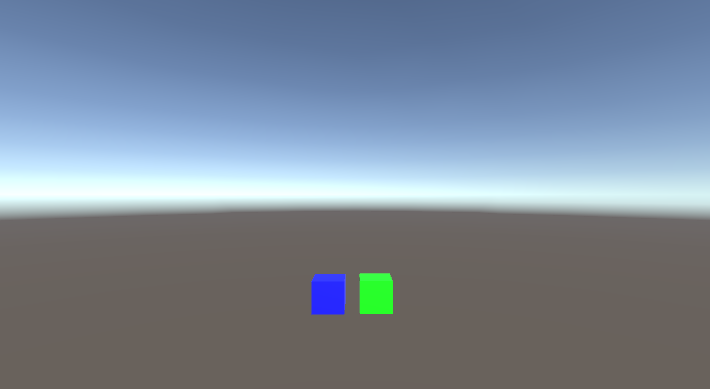
}

}

Вигляд сцени після її запуску:

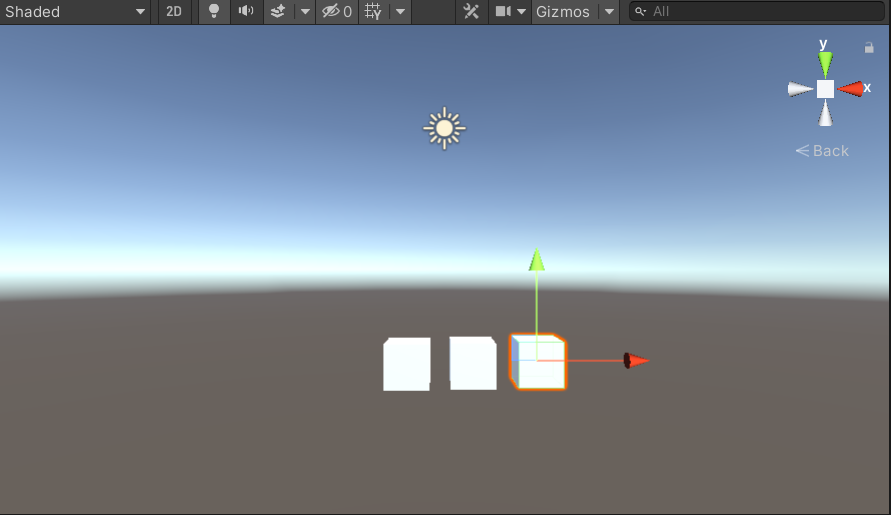


Вигляд сцени під час натискання на один з кубів:

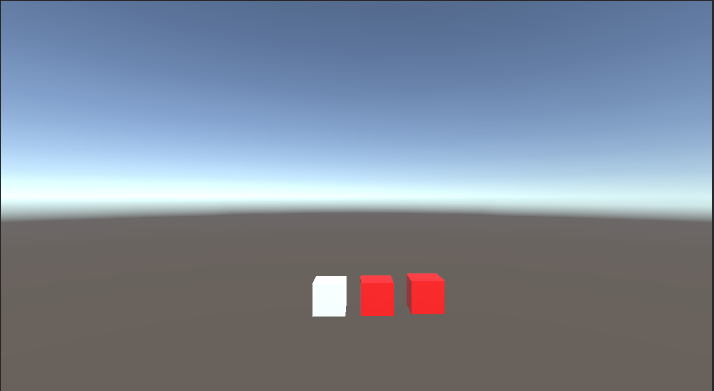


***Завдання 5:*** Створіть копію кубика SingleToneCube та розмістіть його поруч з іншими. Відсліткуйте, як тепер будуть поводити себе тестові кубики.

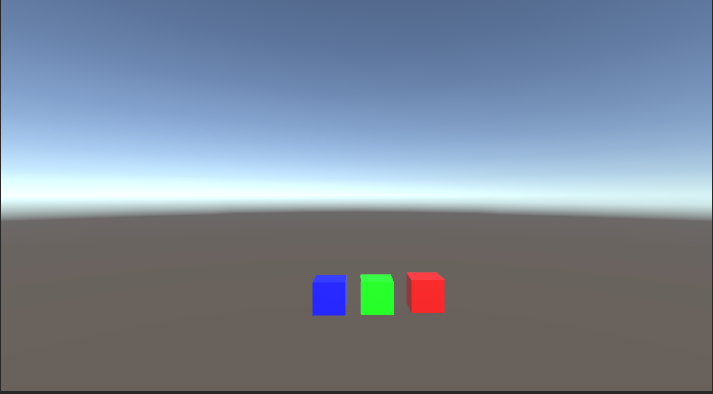
Створена копія кубика SingleToneCube:



Вигляд сцени піcля її запуску:



Вигляд сцени під час натискання на один з кубів:



***Висновки:*** у ході лабораторної роботи ми ознайомились з середовищем Unity та його налаштуваннями. Отримати практичні навички зі створення об’єктів та скриптів, а також ознайомитися з практичним використання шаблону програмування. Познайомились з паттерном «Одинак» та його практичним використанням.

Додаткові завдання

***Завдання 1:*** Модифікуйте клас SingleTone за допомогою массивів або списків так, що би на натискання клавіші миші могли відповідати усі екземпляри цього класу.

Лістинг класу SingleTone:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class SingleToneCube : MonoBehaviour

{

public static SingleToneCube Instance { get; private set; }

private static List<SingleToneCube> \_singleToneCubesList = new List<SingleToneCube>();

private MeshRenderer \_renderer;

void Start()

{

\_singleToneCubesList.Add(this);

\_renderer = GetComponent<MeshRenderer>();

SetColorState(false);

Instance = this;

}

public void SetColorState(bool state)

{

foreach (var cube in \_singleToneCubesList)

{

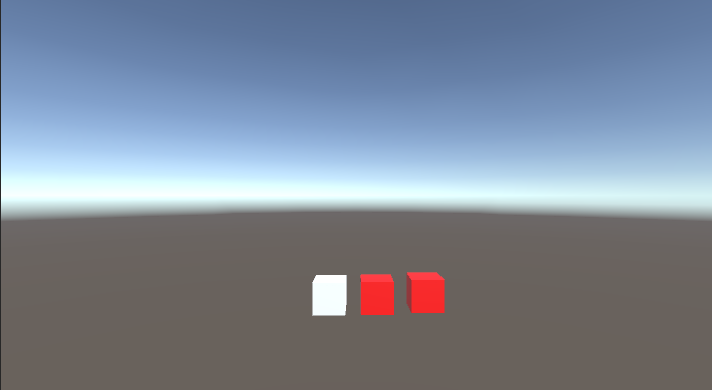
cube.\_renderer.material.color = (state) ? Color.green : Color.red;

}

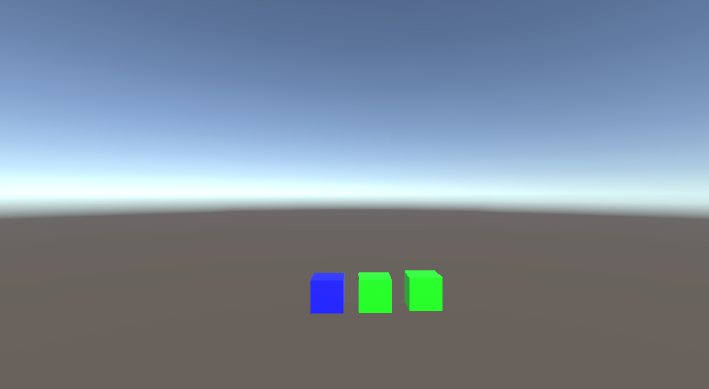
}

}

Вигляд сцени піcля її запуску:



Вигляд сцени під час натискання на один з кубів:



***Завдання 2:*** Створіть окремий клас, що реалізує шаблон «Одинак» і залежить від класу SingleToneCube.

Лістинг новоствореного власного класу MyCube з шаблоном «Одинак»:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class MyCube : MonoBehaviour

{

private MyCube() { }

public static MyCube Instance { get; private set; }

private MeshRenderer \_renderer;

void Start()

{

\_renderer = GetComponent<MeshRenderer>();

Instance = this;

}

public void SetColorState(bool state)

{

\_renderer.material.color = state ? Color.grey : Color.cyan;

}

}

Лістинг класу SingleToneCube:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class SingleToneCube : MonoBehaviour

{

public static SingleToneCube Instance { get; private set; }

private static List<SingleToneCube> \_singleToneCubesList = new List<SingleToneCube>();

private MeshRenderer \_renderer;

void Start()

{

\_singleToneCubesList.Add(this);

\_renderer = GetComponent<MeshRenderer>();

SetColorState(false);

Instance = this;

}

public void SetColorState(bool state)

{

foreach (var cube in \_singleToneCubesList)

{

cube.\_renderer.material.color = (state) ? Color.green : Color.red;

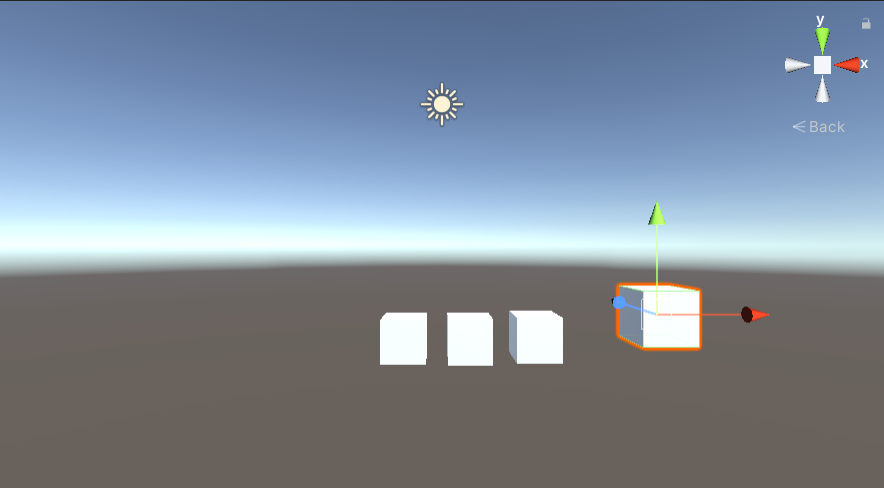
}

MyCube.Instance.SetColorState(state);

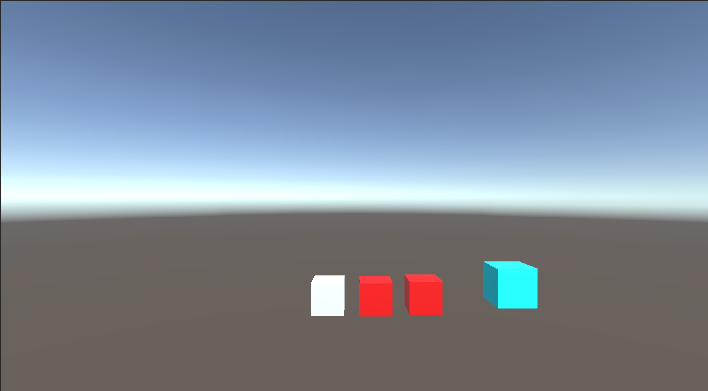
}

}

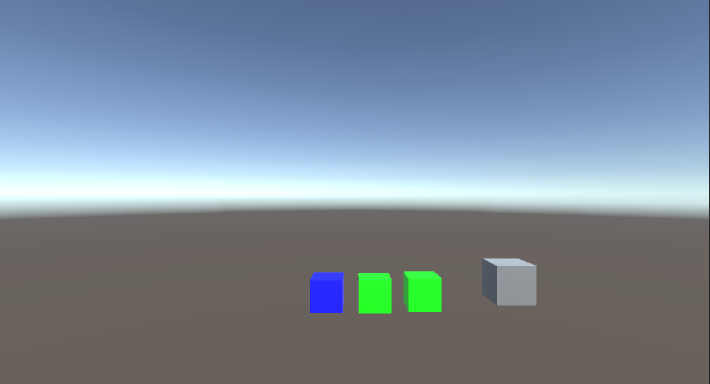
Створення нового куба MyCube:



Вигляд сцени піcля її запуску:



Вигляд сцени під час натискання на один з кубів:



Відповіді на запитання

1. Що таке паттерн програмування, для чого він слугує?

Паттерн програмування — це прийом в програмуванні, що описує загальне рішення проблеми проектування, що повторюється неодноразово у багатьох проектах. Розробники програмного забезпечення адаптують шаблонне рішення до свого конкретного проекту. Патерни програмування допомагають розв’язати задачі та зробити це використовуючи більш оптимальний процес. Це допомагає зробити код більш зручним у написанні та уникнути багатьох можливих помилок.

1. Що таке ігровий рушій?

Ігровий рушій – це середовище розробки програмного забезпечення в основному для створення відеоігор. Інші варіанти цього галузевого терміну – це ігрова архітектура або ігровий фреймворк. Основні функції ігрових механізмів часто включають 2D або 3D-графіку, фізичний движок, анімацію, штучний інтелект, звук та потокове передавання.

1. Яким чином поводить себе класична реалізація паттерну «Одинак», коли існує декілька екземплярів класу із цим паттерном?

Класична реалізація паттерну «Одинак» є без захисту від перезапису і через це, якщо ми створюємо ще один новий клас до нашого вже попередньо реалізованого в «Одинаку», то у цей новий клас просто запишеться старий клас. Якщо ж в «Одинаку» не було класу, то новий клас стане першим класом.

1. Чому, на вашу думку, паттерн одинак називають «антипаттерном»?

Проблема цього паттерну може бути в тестуванні. «Одинак» передбачає програму глобальний стан, тому ви не можете просто взяти і ізолювати класи, які покладаються на Singleton. Тому, якщо ви хочете протестувати якийсь клас, то ви зобов'язані разом з ним тестувати і Singleton. Стан «Одинаки» може змінюватися, що не дозволяє запускати декілька тестів паралельно і декілька викликів одного й того самого тесту можуть привести до різних результатів. В тему глобального стану також можна додати, що він є неприпустимим в ООП, що несе за собою його зрушення в структурно-процедурному підході.

За приклад можемо взяти звичайну базу даних. Вона є незмінним способом отримання і надходження нової інформації. База даних є глобальною для сервера, який у свою чергу надсилає запити, отримує данні, змінює їх.

Якщо брати приклад з життя, то це може бути модель роутера в офісі, яким буде користуватись безліч працівників. При цьому, якщо з’явиться новий робітник, то йому просто потрібно буде дізнатися пароль, а не купляти собі окремий роутер.