федеральное государственное автономное образовательное учреждение высшего образования

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
(ВЫСШАЯ ШКОЛА ПЕЧАТИ И МЕДИАИНДУСТРИИ)
(Факультет информационных технологий)

(Институт Принтмедиа и информационных технологий)
Кафедра Информатики и информационных технологий

направление подготовки
09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № 3

Дисциплина: Шаблоны проектирования

Тема: Система игровых событий

Выполнил(а): студент(ка) группы 221-3711

Морозов К.А.
(Фамилия И.О.)

Дата, подпись _____
(Дата) (Подпись)

Проверил: _____
_____
(Фамилия И.О., степень, звание) **(Оценка)**

Дата, подпись _____ _____
(Дата) (Подпись)

Замечания:

_____

_____

Москва 2024

Лабораторная работа №3

Управление игровыми активами

Цель: Разработать систему управления активами (например, управление различными скинами или оружием) с использованием шаблона Приспособленец (Flyweight). Его еще называют “Легковесом”.

Описание: В играх часто используется множество активов, таких как скины, оружие, и т.д. Шаблон Приспособленец позволяет эффективно управлять и рендерить множество игровых активов, минимизируя потребление памяти.

Шаги:

Определение активов:
- Выберите набор игровых активов, которые вы хотите управлять, например, разные скины, оружие или другие игровые предметы.

Реализация шаблона Flyweight:
- Создайте "легковесные" объекты для каждого актива. Эти объекты должны содержать общую информацию о активе, а уникальные детали (например, текстура или цвет) должны передаваться отдельно при рендеринге.
- Разработайте механизм для управления этими "легковесными" объектами, обеспечивая быстрый доступ и минимальное потребление памяти.

Тестирование:
- Запустите вашу игру и проверьте, как система управляет и рендерит игровые активы. Оцените производительность и потребление памяти.

Листинг:

1. SkinManager

- **Поле skinFactory**: Хранит ссылку на фабрику скинов (**SkinFactory**).
- **Метод Start**: Создает экземпляр **SkinFactory** и применяет скин к текущему объекту.
- **Метод ApplySkinToCharacter (public)**: Принимает ключ скина и объект персонажа. Использует фабрику для получения скина по ключу и применяет его к указанному объекту.

```
using UnityEngine;

public class SkinManager : MonoBehaviour
{
    private SkinFactory skinFactory;

    void Start()
    {
        skinFactory = new SkinFactory();

        ApplySkinToCharacter("Skin1", this.gameObject);
    }

    public void ApplySkinToCharacter(string skinKey, GameObject character) // Изменено на public
    {
        IFlyweightSkin skin = skinFactory.GetSkin(skinKey);
        if (skin != null)
        {
            skin.ApplySkin(character);
        }
        else
        {
            Debug.LogError("Skin not found: " + skinKey);
        }
    }
}

```

2. SkinFactory

- **Поле skins**: Хранит кэш скинов (словарь ключей и объектов скинов).
- **Метод GetSkin**: Возвращает скин по ключу из кэша или загружает новый скин из ресурсов, добавляя его в кэш.

```
using System.Collections.Generic;
using UnityEngine;

public class SkinFactory
{
```

```
    private Dictionary<string, IFlyweightSkin> skins = new Dictionary<string, IFlyweightSkin>();

    public IFlyweightSkin GetSkin(string key)
    {
        if (skins.ContainsKey(key))
        {
            return skins[key];
        }

        // Загружаем ресурс скин по ключу
        Sprite sprite = Resources.Load<Sprite>(key);
        if (sprite != null)
        {
            IFlyweightSkin skin = new FlyweightSkin(sprite);
            skins[key] = skin;
            return skin;
        }

        return null;
    }
}
```

3. IFlyweightSkin

- **Интерфейс**: Объявляет метод **ApplySkin**, который должен быть реализован всеми скинами.

```
using UnityEngine;

public interface IFlyweightSkin
{
    void ApplySkin(GameObject character);
}
```

4. FlyweightSkin

- **Поле skinSprite**: Хранит спрайт скина.
- **Конструктор**: Принимает спрайт и сохраняет его в поле **skinSprite**.
- **Метод ApplySkin**: Применяет спрайт к компоненту **SpriteRenderer** объекта персонажа.

```
using UnityEngine;

public class FlyweightSkin : IFlyweightSkin
{
    private Sprite skinSprite;

    public FlyweightSkin(Sprite sprite)
    {
        this.skinSprite = sprite;
```

```
    }

    public void ApplySkin(GameObject character)
    {
        SpriteRenderer renderer = character.GetComponent<SpriteRenderer>();
        if (renderer != null)
        {
            renderer.sprite = skinSprite;
        }
    }
}
```

5. SkinPerformanceTest

- **Поле characterPrefab**: Префаб персонажа, к которому будут применяться скины.
- **Поле characterCount**: Количество персонажей для теста (1000).
- **Поле skinManager**: Ссылка на объект **SkinManager**.
- **Метод Start**:
  - Создает экземпляр **SkinManager**.
  - Создает массив для хранения ссылок на 1000 объектов персонажей.
  - Создает 1000 экземпляров персонажа и сохраняет их в массиве.
  - Использует **Stopwatch** для измерения времени, затраченного на применение скинов ко всем персонажам.
  - Применяет скин ко всем персонажам и выводит затраченное время в консоль.

```
using UnityEngine;
using System.Diagnostics; // Не забудьте подключить пространство имен System.Diagnostics для использования Stopwatch

public class SkinPerformanceTest : MonoBehaviour
{
    public GameObject characterPrefab; // Префаб персонажа
    public int characterCount = 10000; // Количество персонажей для теста
    private SkinManager skinManager;

    void Start()
    {
        // Создаем объект SkinManager
        GameObject skinManagerObject = new GameObject("SkinManager");
        skinManager = skinManagerObject.AddComponent<SkinManager>();

        // Создаем массив для хранения ссылок на созданные объекты
        GameObject[] characters = new GameObject[characterCount];

```

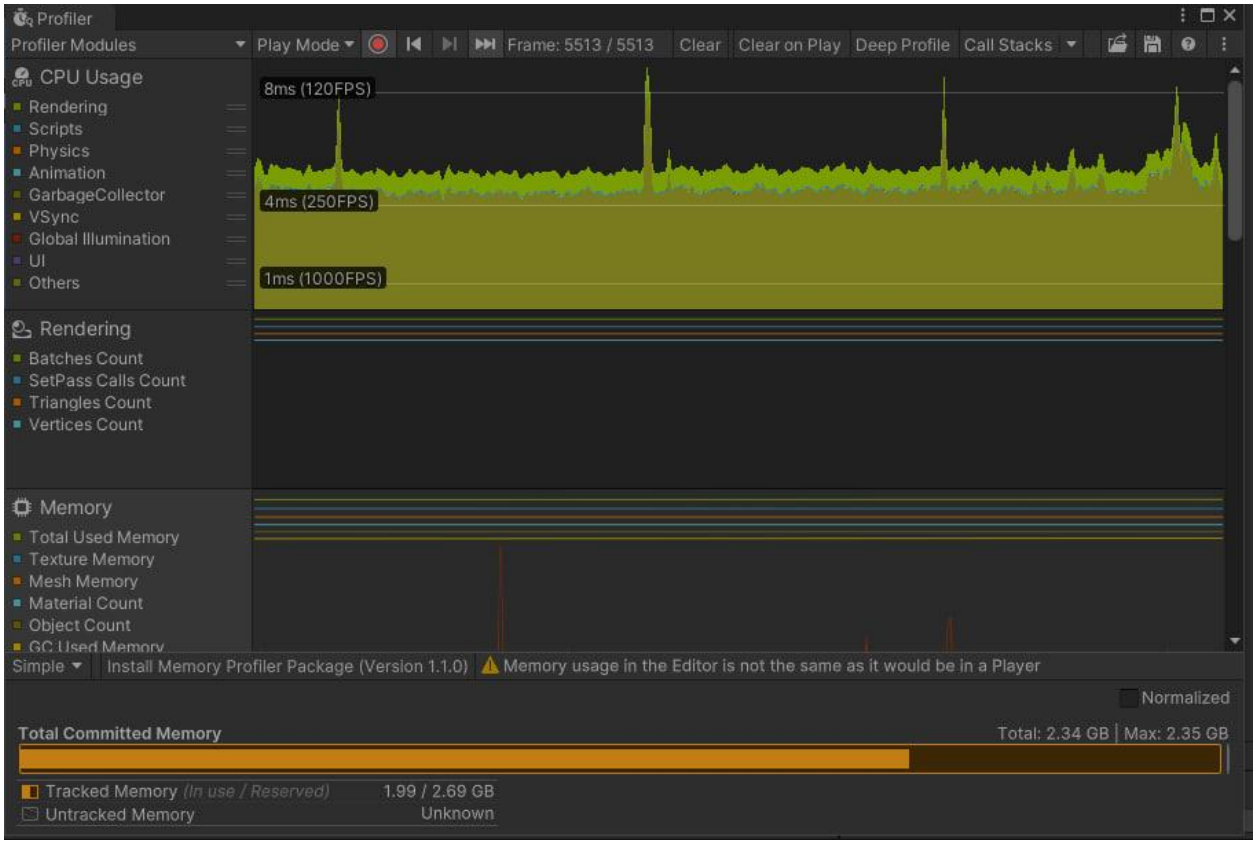
```
        // Создаем персонажей и добавляем их в массив
        for (int i = 0; i < characterCount; i++)
        {
            characters[i] = Instantiate(characterPrefab, new Vector3(i * 1.5f, 0, 0), Quaternion.identity);
        }

        // Создаем и запускаем Stopwatch
        Stopwatch stopwatch = new Stopwatch();
        stopwatch.Start();

        // Применяем скин ко всем персонажам
        for (int i = 0; i < characterCount; i++)
        {
            skinManager.ApplySkinToCharacter("Skin1", characters[i]);
        }

        // Останавливаем Stopwatch и выводим результат
        stopwatch.Stop();
        UnityEngine.Debug.Log("Time taken to apply skin to 1000 characters: " + stopwatch.ElapsedMilliseconds + " ms");
    }
}
```

Ход работы

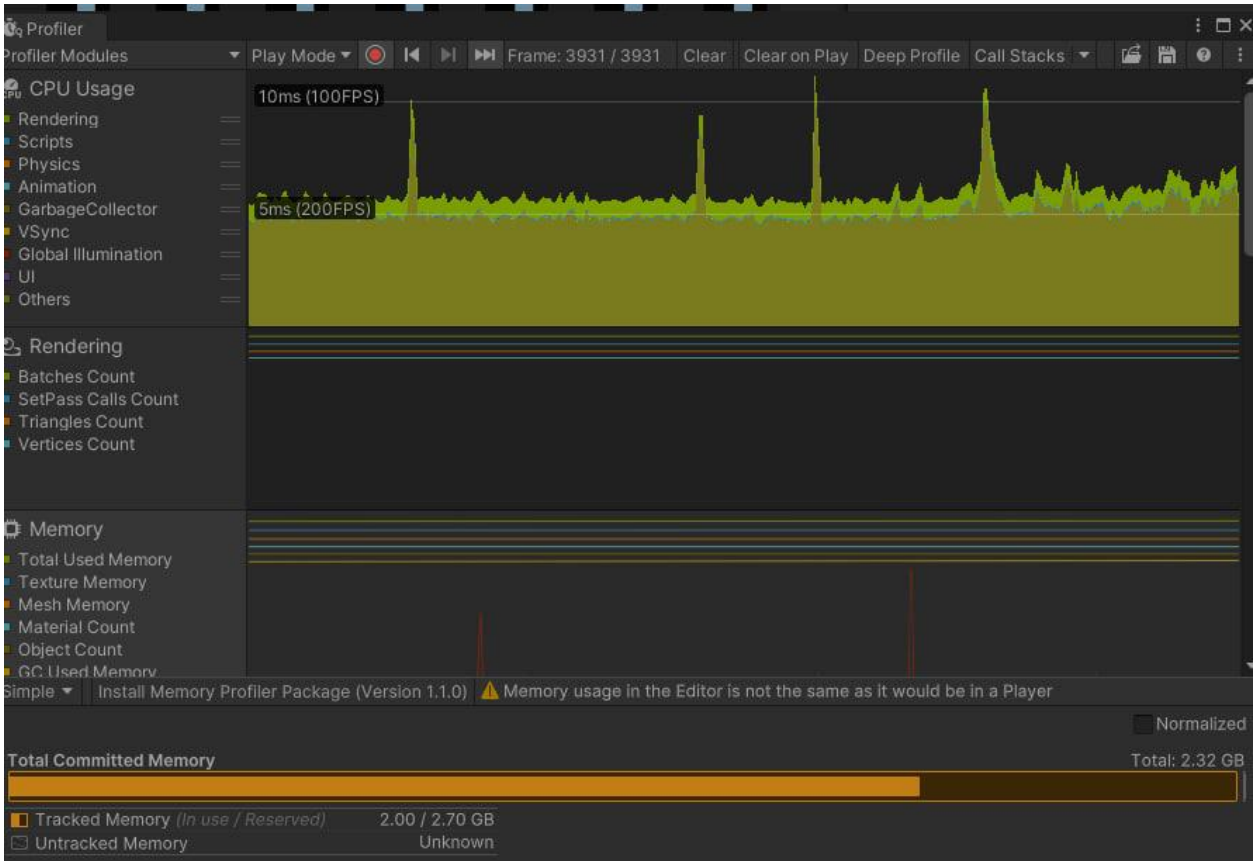
Основные преимущества применения паттерна Flyweight:

- **Снижение потребления памяти**: За счет повторного использования общих частей состояния уменьшается количество создаваемых объектов.
- **Повышение производительности**: Меньшее количество объектов снижает нагрузку на сборщик мусора и улучшает общую производительность приложения.
- **Централизованное управление состоянием**: Фабрика централизует создание и управление общими частями состояния, что упрощает код и делает его более управляемым.

С использованием:



Без:



Ссылка на проект в Github: https://github.com/Sollimba/Hablon2