ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки
09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № 2

Дисциплина: «Шаблоны проектирования»

Тема: *Система игровых событий*

Выполнил: студент группы: 231-339

Карапетян Нвер Каренович
(Фамилия И.О.)

Дата, подпись: 21.03.25 _____
(Дата) (Подпись)

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

Дата, подпись _____ _____
(Дата) (Подпись)

Москва
2025

Цель:

Создайте систему событий, в которой различные игровые объекты могут подписываться и реагировать на игровые события с использованием определенного шаблона проектирования.

Описание:

В играх часто происходят различные события: от перемещения игрока до завершения задания. Для управления такими событиями и реакцией на них различных компонентов игры необходима эффективная система. Использование подходящего шаблона проектирования может сделать эту систему более гибкой и удобной.

План работы

Определение игровых событий:

Определить набор игровых событий, которые должны быть реализованы. Это события, связанные со сбором игровых монеток персонажем и завершением уровня при условии, что игроком были собраны все монетки уровня.

Реализация системы событий:

- Создать «издателя» событий, который будет генерировать события.
- Разработать механизм подписки, который позволит другим игровым объектам «подписываться» на интересующие их события.
- Игровые объекты, «подписанные» на события, должны реагировать на них соответствующим образом.

Тестирование:

Запустить вашу игру и проверить, как различные компоненты реагируют на игровые события в реальном времени.

Ход работы

Паттерн «Наблюдатель» (Observer)

Событийно-ориентированное программирование (Event-Driven Programming) является одной из ключевых парадигм в разработке интерактивных приложений и игр. В данной парадигме **событие** представляет собой сигнал о том, что произошло определённое действие (например, сбор монеты, нажатие кнопки или завершение уровня); **издатель** события генерирует событие в момент наступления определённых условий; **подписчики** представляют из себя объекты, которые «слушают» события и реагируют на них соответствующим образом.

Паттерн «Наблюдатель» является одним из наиболее распространённых шаблонов проектирования, позволяющих реализовать событийную модель.

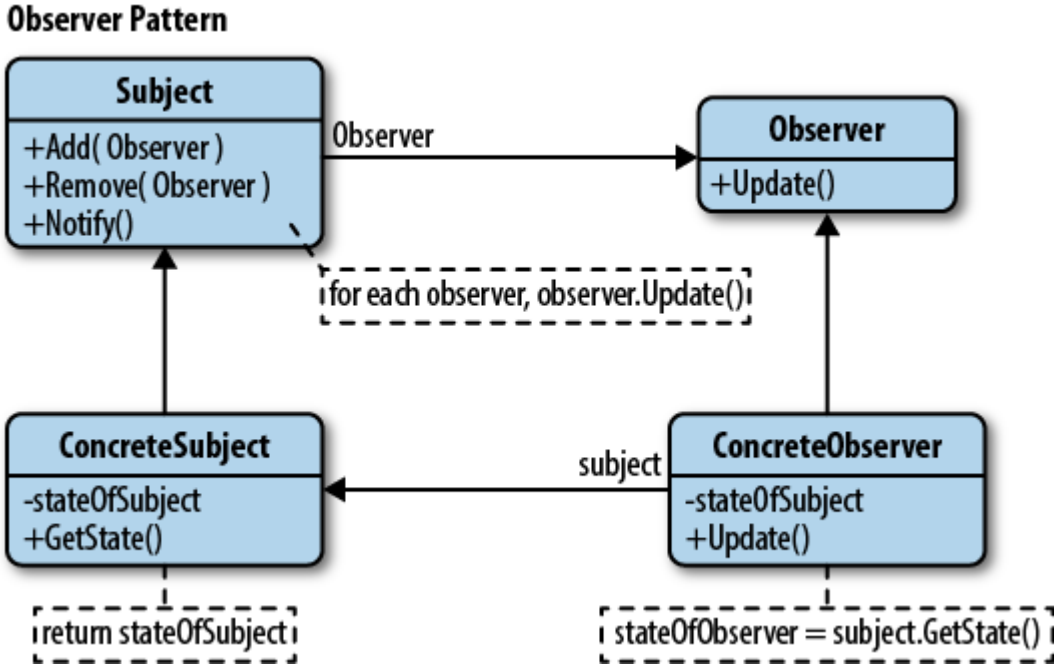

Рисунок 1. Схема паттерна «Наблюдатель».

Шаблон Observer определяет зависимость «один-ко-многим» между объектами так, что при изменении состояния одного объекта все зависящие от него объекты уведомляются и обновляются автоматически. Паттерн Observer инкапсулирует главный (независимый) компонент в абстракцию Subject и изменяемые (зависимые) компоненты в иерархию Observer. Шаблон Observer также определяет часть «View» в модели Model-View-Controller (MVC).

В реализации на C# широко используются события и делегаты, что позволяет легко организовать подписку и отписку от событий.

Система событий (EventManager)

Создан класс `EventManager`, реализующий паттерн «Наблюдатель». Он содержит события `OnCoinCollected` с передачей значения собранной монеты и `OnLevelCompleted` для оповещения об окончании уровня:

Листинг 1. Класс EventManager.

```
public class EventManager : MonoBehaviour
{
    public static EventManager Instance { get; private set; }

    private void Awake()
    {
        if (Instance is null)
            Instance = this;
        else
            Destroy(gameObject);
    }

    public event Action<int> OnCoinCollected;
    public void CoinCollected(int value) => OnCoinCollected?.Invoke(value);

    public event Action OnLevelCompleted;
    public void LevelCompleted() => OnLevelCompleted?.Invoke();
}
```

Монета при столкновении с игроком отправляет событие через `EventManager`:

Листинг 2. Класс Coin.

```
public class Coin : MonoBehaviour
{
    [SerializeField] private int value;
    private bool hasTriggered;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Player") && !hasTriggered)
        {
            hasTriggered = true;

            EventManager.Instance.CoinCollected(value);
            Destroy(gameObject);
```

```
        }
    }
}
```

`CoinManager` подписывается на событие `OnCoinCollected` и обновляет счетчик монет:

Листинг 3. Класс CoinManager.

```
public class CoinManager : MonoBehaviour
{
    public static CoinManager instance;
    private int coins;
    [SerializeField] private TMP_Text coinsDisplay;

    private void Awake()
    {
        if (instance is null)
            instance = this;
        else
            Destroy(gameObject);
    }

    private void OnEnable() => EventManager.Instance.OnCoinCollected +=
ChangeCoins;

    private void OnDisable() => EventManager.Instance.OnCoinCollected -=
ChangeCoins;

    private void ChangeCoins(int amount)
    {
        coins += amount;
        UpdateDisplay();
    }

    private void UpdateDisplay()
    {
        if (coinsDisplay is not null)
            coinsDisplay.text = coins.ToString();
    }

    public int GetCoins() => coins;
}
```

При входе игрока в зону финиша проверяется количество собранных монет, и если условие выполнено — вызывается событие завершения уровня:

Листинг 4. Класс FinishTrigger.

```
public class FinishTrigger : MonoBehaviour
{
    [SerializeField] private int requiredCoins = 7;
    [SerializeField] private TMP_Text messageText;
    [SerializeField] private float messageDuration = 2f;
    private bool levelCompleted = false;

    private void Start()
    {
        if (messageText is not null)
            messageText.gameObject.SetActive(false);
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Player") && !levelCompleted)
        {
            int collectedCoins = CoinManager.instance.GetCoins();

            if (collectedCoins >= requiredCoins)
            {
                levelCompleted = true;
                PlayerPrefs.SetInt("Level1Completed", 1);
                PlayerPrefs.Save();

                EventManager.Instance.LevelCompleted();
                SceneManager.LoadScene("LevelMenu");
            }
            else
            {
                StartCoroutine(ShowMessage($"Собрано {collectedCoins} из {required-
Coins} монеток!"));
            }
        }
    }

    private IEnumerator ShowMessage(string message)
    {
        if (messageText is not null)
        {
            messageText.text = message;
            messageText.gameObject.SetActive(true);
            yield return new WaitForSeconds(messageDuration);
            messageText.gameObject.SetActive(false);
        }
    }
}
```

Таким образом система сбора монет и завершения уровня работает через события: монеты отправляют уведомления, а подсистема монет обновляет UI, а система завершения уровня проверяет условия и инициирует переход в меню уровней:

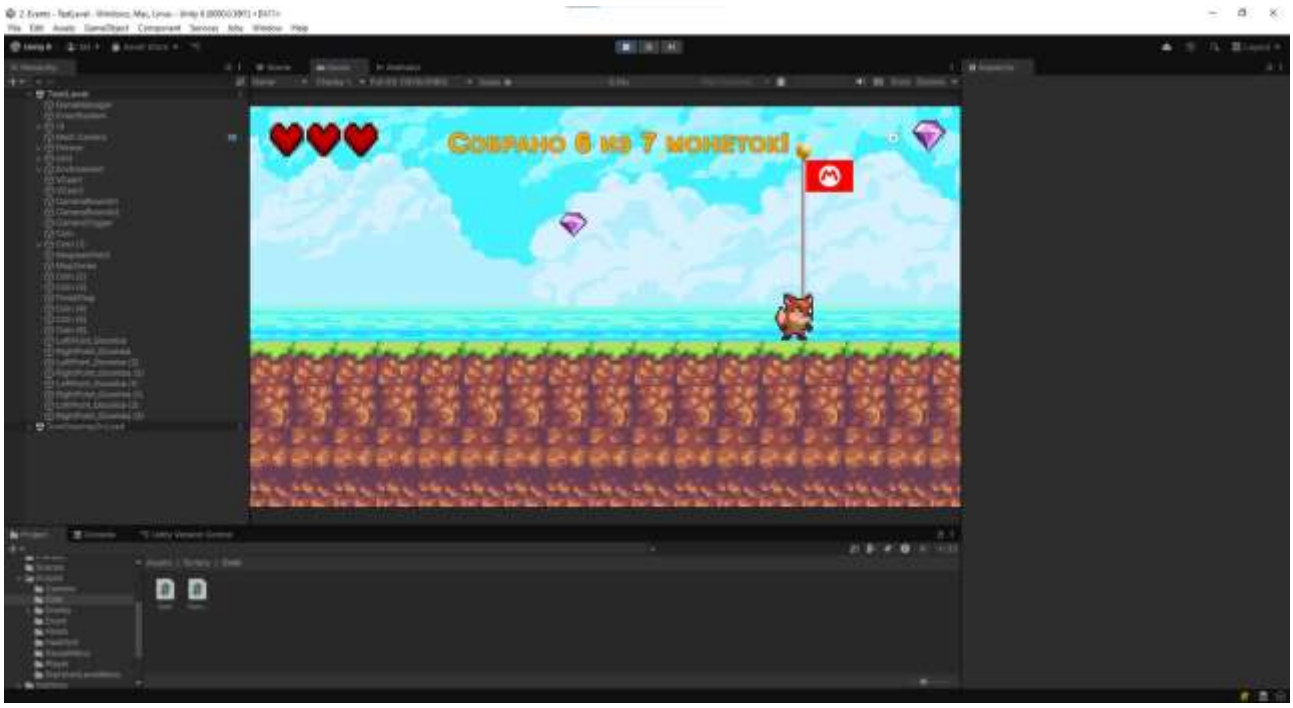

Рисунок 2. Результат работы.