Міністерство освіти і науки України
Національний університет «Львівська політехніка»

ЗВІТ

Про виконання лабораторної роботи № 3

На тему: “*Твірні шаблони*”
з дисципліни «Моделювання та аналіз ПЗ»

Лектор:
доцент каф. ПЗ
Сердюк П.В.

Виконав:
студ. гр. ПЗ-22
Місяйло О.О.

Прийняв:
Викл. каф.ПЗ
Микуляк А.В.

«___» _____ 2023 р.

∑ = ____ _____

Львів – 2023

Тема: Твірні шаблони.
Мета: Здобути навички використання твірних шаблонів проектування при моделюванні програмних систем.

Теоретичні відомості

Твірні шаблони (англ. Creational patterns) — це шаблони проєктування, що абстрагують процес побудови об'єктів. Вони допоможуть зробити систему незалежною від способу створення, композиції та представлення її об'єктів.

Шаблон, який породжує класи використовує успадкування, щоб варіювати створюваний клас, а шаблон, що створює об'єкти,делегує інстанціювання іншому об'єктові.

Ці шаблони важливі, коли система більше залежить від композиції об'єктів, ніж від успадкування класів.

Таким чином, замість прямого кодування фіксованого набору поведінок, визначається невеликий набір фундаментальних поведінок, за допомогою композиції яких можна отримувати складніші. Тобто для створення об'єктів з конкретною поведінкою потрібно щось більше, ніж просте інстанціювання екземпляру класу.

Шаблони, що породжують, інкапсулюють знання про конкретні класи, які застосовуються у системі та приховують деталі того, як ці класи створюються і стикуються між собою.

Єдина інформація про об'єкти, що відома системі — їхні інтерфейси.

Приклади твірних шаблонів:

- Шаблон Одинак (Singleton)
- Шаблон Будівельник (Builder)
- Шаблон Абстрактна фабрика (Abstract Factory)
- Шаблон Прототип (Prototype)

Завдання

Розробити твірні шаблони проектування відповідно до прецедентів обраного варіанту ігрової логіки. Вибрати один з прецедентів, для якого найбільш доцільно застосувати твірний шаблон (не всі прецеденти цього потребуватимуть).

Обов’язкові шаблони для реалізації:

- Singleton
- Абстрактна фабрика

Реалізувати один із шаблонів на вибір:

- Будівельник
- Прототип

Представити діаграму класів ігрового проекту (із відображенням на ній використаних шаблонів).

**Реалізувати шаблони Singleton для класів, які потребують лише один екземпляр об’єкту в моделі (не в UI). Це може бути карта, основний рухомий об’єкт, екземпляр цілої гри тощо.*

Абстрактна Фабрика може реалізовувати різні варіанти складності гри, стилів гри, підтримку різних розширень екрану за допомогою шаблону, яка б створювала різні набори компонентів при різних розширення.

Будівельник запотребований для побудови карт гри чи інших складних об'єктів, яке містить різні компоненти, які по різному групуються при різних розширеннях. Наприклад, вікно карти міститиме 10x10 комірок при малих розширеннях і 20x20 комірок при великих розширеннях. При великих розширеннях, також може містити додаткові панелі, які відсутні при малих розширеннях екрану і т.д.

Прототип запотребований для масової ініціалізації подібних об’єктів. Наприклад є класс Unit, який має малюнок, HP та інші характеристики. Можна реалізувати декілька реалізацій цього класу, наприклад: Dragon, Wolf і т.д. Кожному з цих класів присвоїти їх відповідні характеристики (малюнки, HP і т.д.), а пізніше розмножувати ці об’єкти шляхом копіювання прототипів.

Хід виконання

1. Першим чином я скачав програму Unity, на якій і буду розробляти свою гру. Накинувши певну карту, я почав добавляти моделі та анімацію. Згодом перейшов до впровадження у проект шаблону Singleton.

```
public class Singleton<T> : MonoBehaviour where T:Component
{
    private static T instance;

    public static T Instance
    {
        get
        {
            if(instance==null)
            {
                instance = FindObjectOfType<T>();

                if (instance == null)
                {
                    GameObject gameObject = new GameObject("Component");
                    instance = gameObject.AddComponent<T>();
                }
            }

            return instance;
        }
    }

    private void Awake()
    {
        if (instance == null)
        {
            instance = this as T;
        }
        else if(instance!= this)
        {
            Destroy(gameObject);
        }
    }
}
```

Рисунок 1

```
public class HeroParameters : Singleton<HeroParameters>
{
    public float speed;
    public float jump;

}
```

Рисунок 2

```
void Update()
{
    speed = HeroParameters.Instance.speed;
    jump = HeroParameters.Instance.jump;
    Move = Input.GetAxis("Horizontal");
    animator.SetFloat("Speed", Mathf.Abs(Move));
    rb.velocity = new Vector2(speed*Move, rb.velocity.y);


    //rest or function code

}
```

Рисунок 3

2. Після реалізації сінглота я перейшов до реалізації фабрики, яка в мене генерує різні об’єкти з Layer “Enemy”;

```
public interface IEnemyFactory
{
    GameObject GetEnemy(String name);
}
```

Рисунок 4

```
public class EnemyFactory : IEnemyFactory
{
    private GameObject EnemySkull;
    private GameObject EnemyCrab;

    public EnemyFactory(GameObject enemyCrab, GameObject enemySkull)
    {
        EnemySkull = enemySkull;
        EnemyCrab = enemyCrab;
    }
    public GameObject GetEnemy(String name)
    {

        if (name == "Crab") return EnemyCrab;
        if (name == "Skull") return EnemySkull;
        return null;
    }

}
```

Рисунок 5

```
public GameObject EnemySkull;
public GameObject EnemyCrab;
private EnemyFactory _enemyFactory;
private string[] enemies = new[] { "Crab", "Skull" };

public float spawnRate = 5;

private float timer = 0;

// Start is called before the first frame update
void Start()
{
    _enemyFactory = new EnemyFactory(EnemyCrab, EnemySkull);
}

// Update is called once per frame
void Update()
{
    if (timer < spawnRate)
    {
        timer += Time.deltaTime;
    }
    else
    {

        GameObject enemy = _enemyFactory.GetEnemy(enemies[Random.Range(0,
enemies.Length)]);
        Instantiate(enemy, transform.position , transform.rotation);
        timer = 0;
    }
}
```

Рисунок 6

3. Останній реалізований шаблон – prototype.

```
public abstract class Prototype : MonoBehaviour
{
    public abstract GameObject clone();
}
```

Рисунок 7

```
public class FireballPrototype : Prototype
{
    public GameObject fireballPrefab;
    public override GameObject clone()
    {
        return fireballPrefab;
    }
}
```

Рисунок 8

```
public class FireballSpawner : MonoBehaviour
{

    public float spawnRate = 15;

    public FireballPrototype _fireballPrototype;
    private float timer;

    // Update is called once per frame
    void Update()
    {
        if (timer < spawnRate)
        {
            timer += Time.deltaTime;
        }
        else
        {
            Instantiate(_fireballPrototype.clone(), transform.position ,
transform.rotation);
            timer = 0;
        }
    }
}
```

Рисунок 9

4. Побудова UML-діаграм

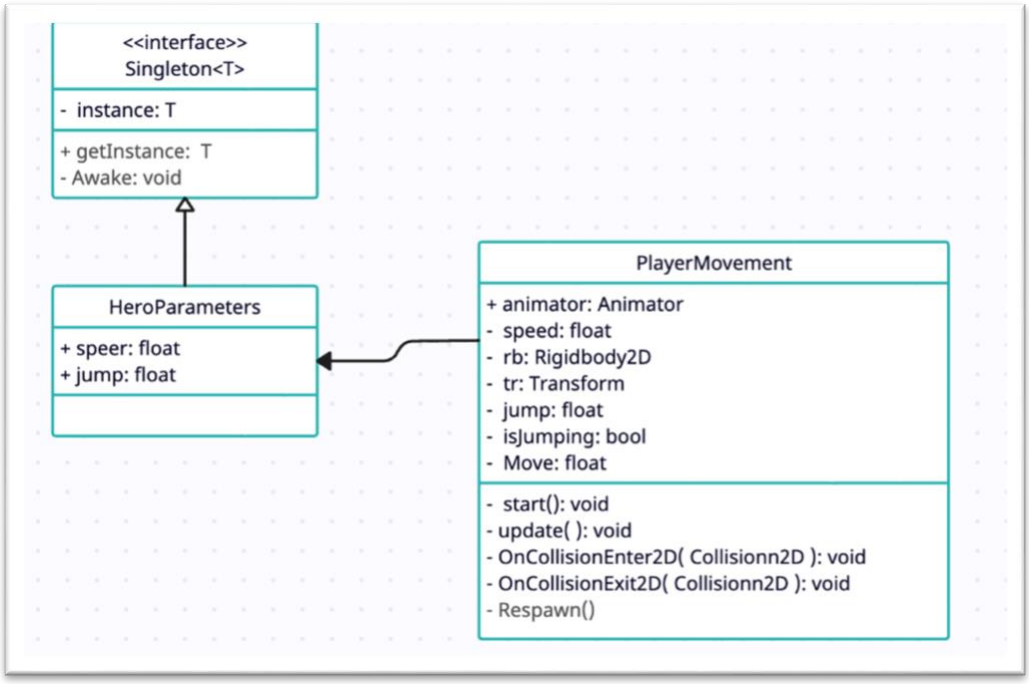

Рисунок 10. Класи, пов'язані з патерном singleton.

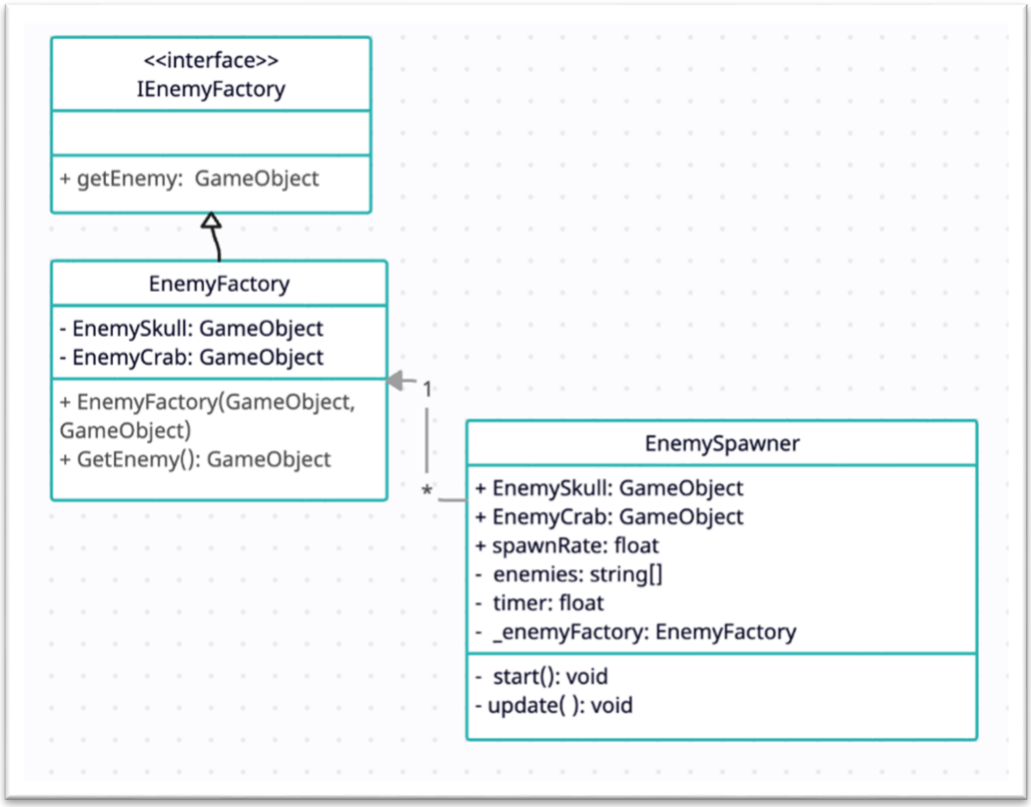

Рисунок 11. Класи, пов'язані з абстрактною фабрикою

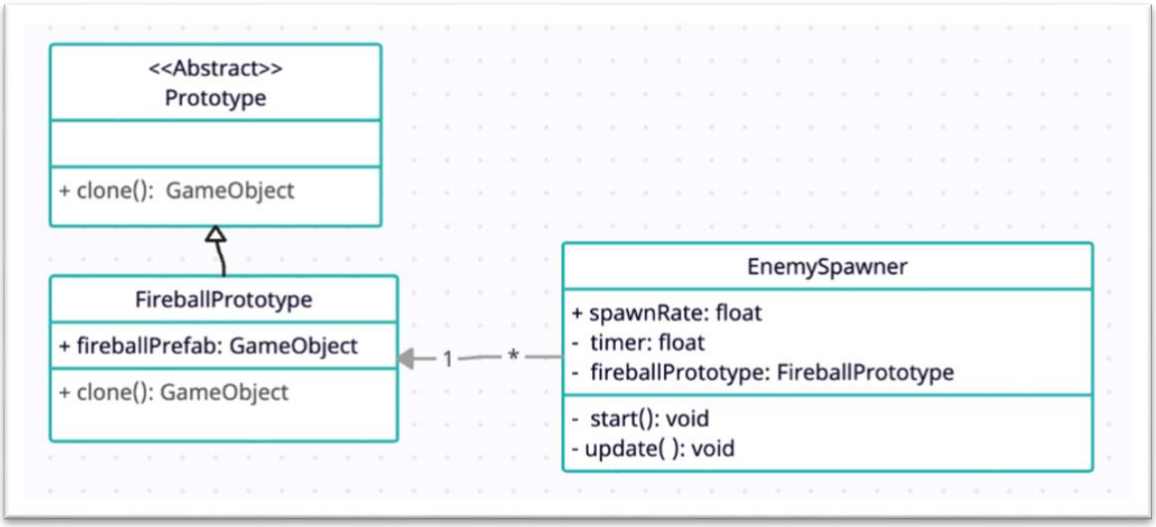

Рисунок 12. Класи, пов'язані з патерном Prototype

Висновок

У цій лабораторній роботі я ознайомився з твірними шаблонами проектування, почав створювати свою гру, у якій застосовував деякі шаблони, а саме: Singleton, Abstract Factory та Prototype.