Flyweight
Proxy

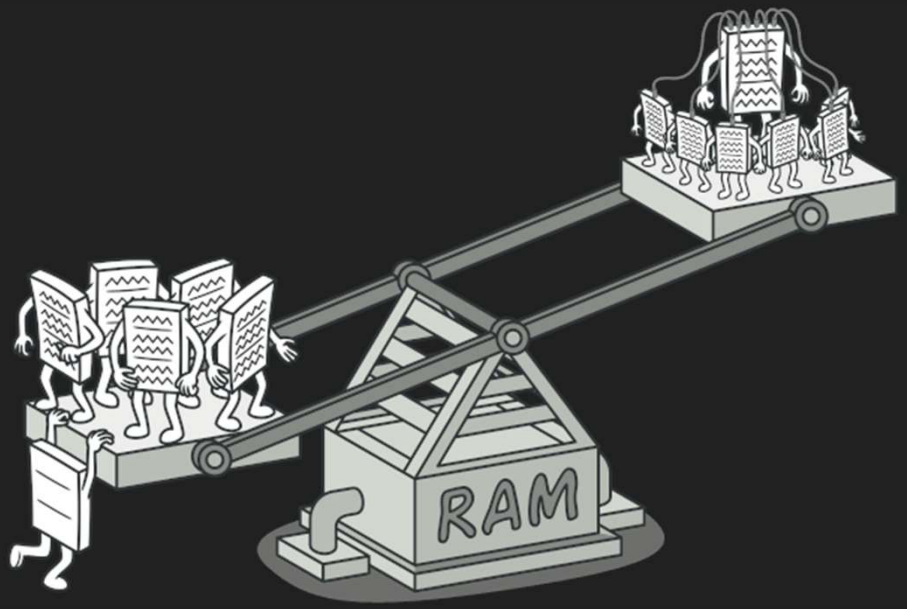
Підготував Цегельник Богдан КН-3

Легковаговик

Також відомий як: **Пристосуванець, Кеш, Flyweight**

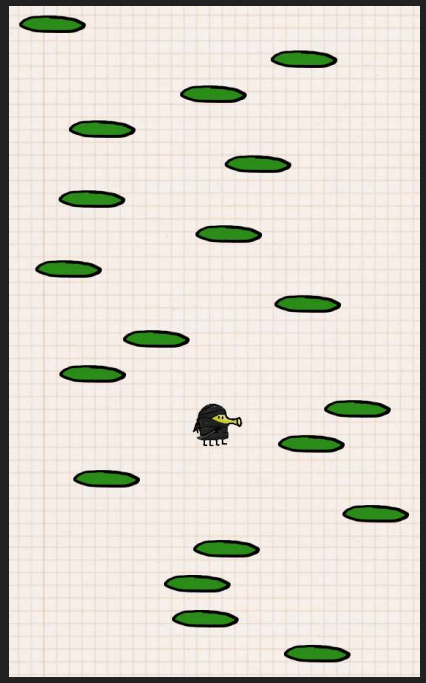
Суть патерну

- Структурний патерн, який допомагає заощадити пам'ять, розподіляючи спільний стан об'єктів між собою, замість зберігання однакових даних у кожному об'єкті.



Розглянемо проблему

- В грі персонаж стрибає нескінченно вгору по платформах, які постійно з'являються над ним. У випадку коли кожна платформа ініціалізуватиме власну текстуру це має ряд проблем:
  - Постійні звернення до диска для завантаження зображення
  - Збільшені витрати пам'яті, оскільки зображення міститься в кожному об'єкт



Можливе рішення

```
class SimplePlatform : public Platform {
    …
    static SDL_Texture *spPlatformTexture;
    void Draw() override;
};
```

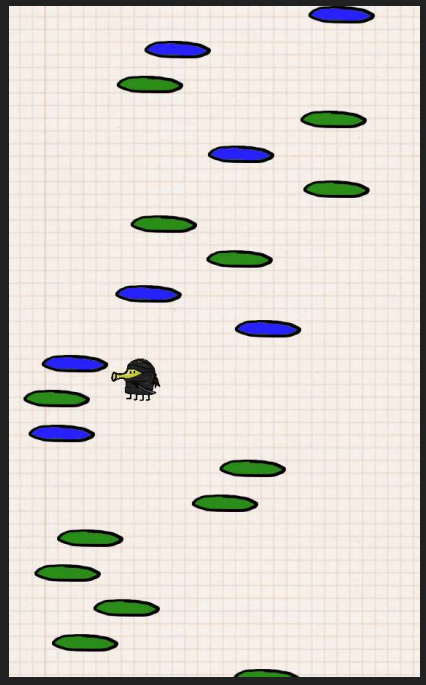
```
SDL_Texture *SimplePlatform::spPlatformTexture = nullptr;


SimplePlatform::SimplePlatform(int x, int y, Renderer& renderer) : Platform(x, y,
renderer) {
        if (SimplePlatform::spPlatformTexture == nullptr)
        {
                // ініціалізація текстури
        }
}


void SimplePlatform::Draw() {
        SDL_RenderCopy(mrRenderer.GetRawRenderer(), spPlatformTexture, NULL,
&mHitBox);
}
```

Нова проблема

- Що робити тоді коли з'явилася потреба у платформах з іншою текстурою?

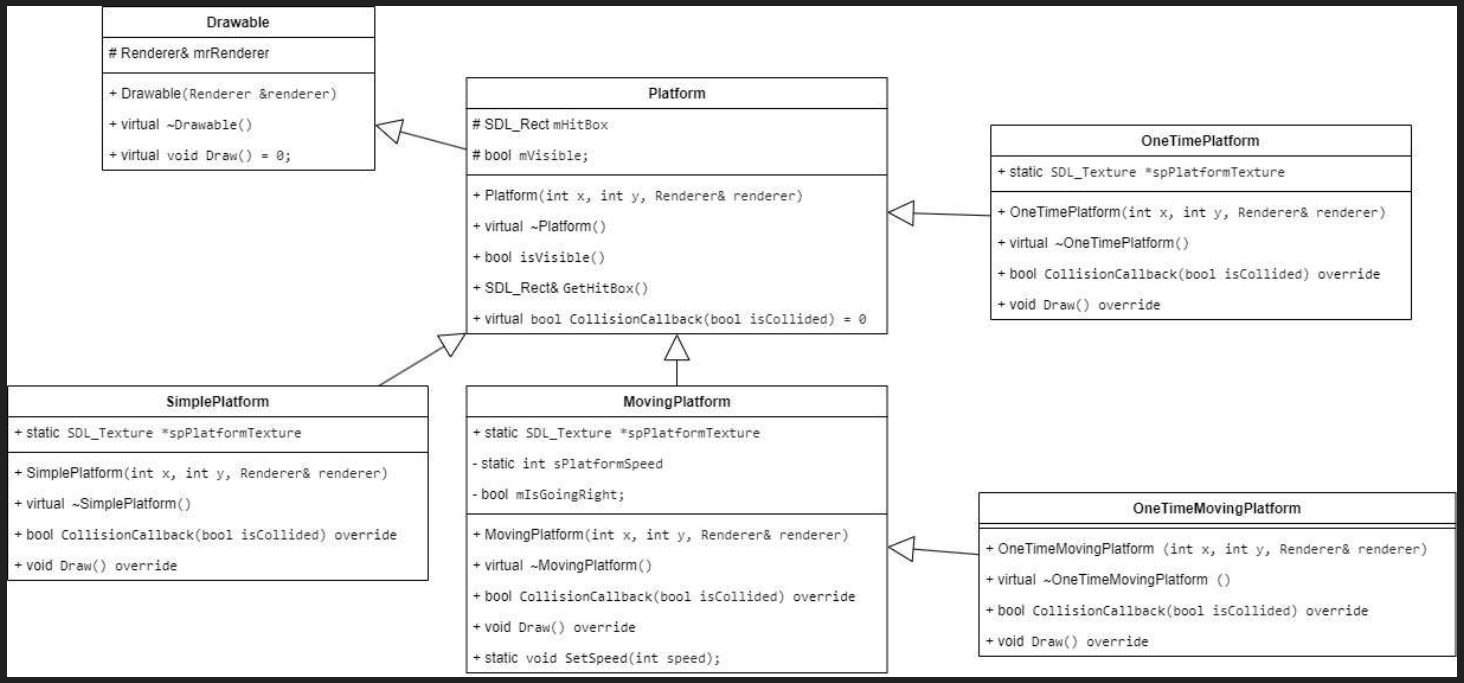


Що зробив я

```
class MovingPlatform : public Platform {
    static SDL_Texture *spPlatformTexture;
    void Draw() override;
};
```

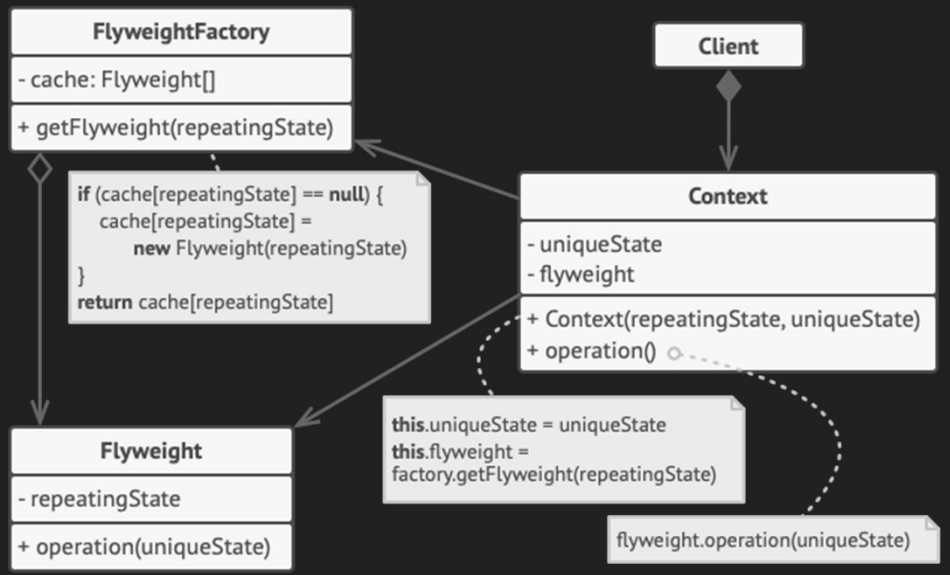
```
class OneTimePlatform : public Platform{
    static SDL_Texture *spPlatformTexture;
    void Draw() override;
};
```

Діаграма класів



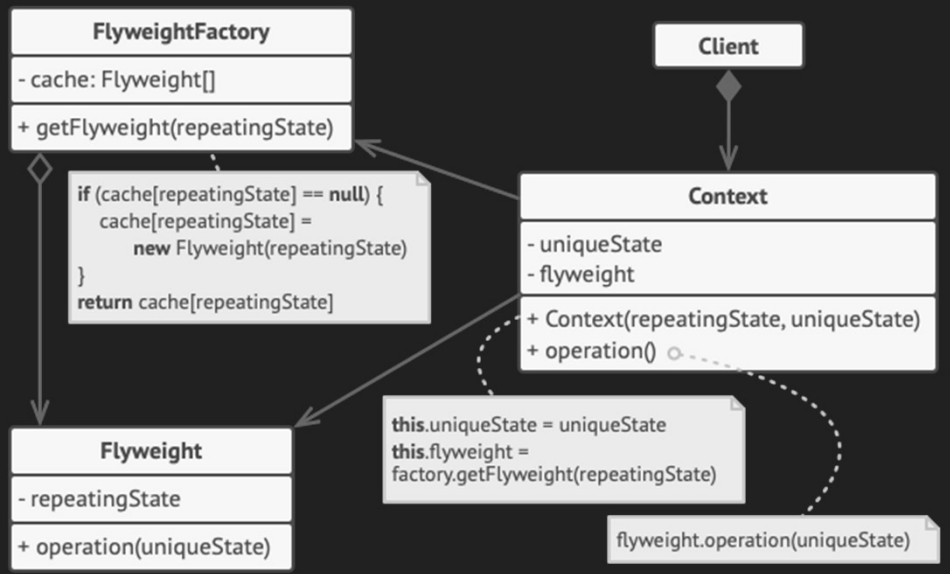
Що пропонує патерн Легковаговик

- Патерн розділяє дані об'єктів на дві частини — легковаговики та контексти.
- **Легковаговик** містить «внутрішній» стан, який повторювався в багатьох первинних об'єктах. Один і той самий легковаговик може використовуватись у зв'язці з безліччю контекстів.
- **Контекст** містить «зовнішню» частину стану, унікальну для кожного об'єкта.

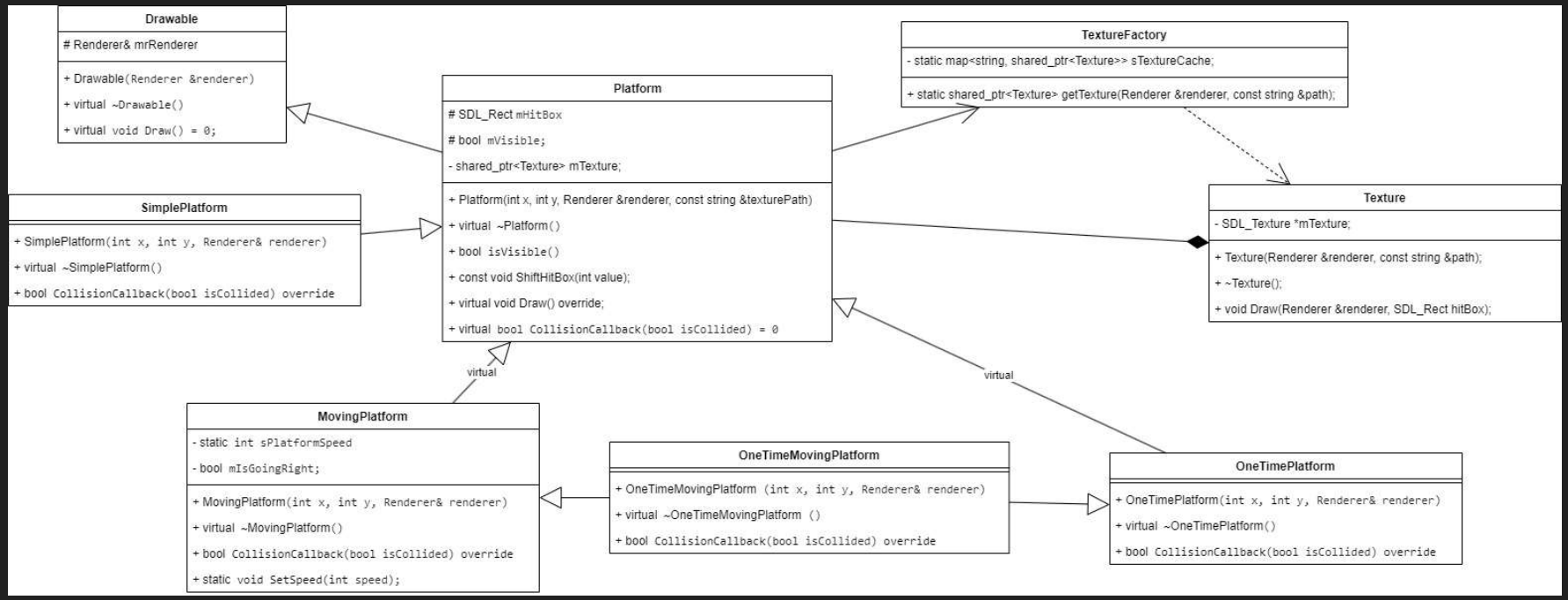


Що пропонує патерн Легковаговик

- Легковаговик є сталим об'єктом, тобто не змінює свій стан створення
- **Фабрика легковаговиків** керує створенням і повторним використанням легковаговиків. Фабрика отримує запити, в яких зазначено бажаний стан легковаговика. Якщо легковаговик з таким станом вже створений, фабрика відразу його повертає, а якщо ні — створює новий об'єкт.



Застосуємо патерн до нашої гри



Переваги та недоліки патерну

Переваги

- Заощаджує оперативну пам'ять
- Зменшує кількість створюваних об'єктів

Недоліки

- Ускладнює програмний код
- Накладає додаткові витрати процесорного часу

Коли ефективно використовувати патерн

- У програмі використовується велика кількість об’єктів
- Через це високі витрати оперативної пам’яті
- Більшу частину стану об’єктів можна винести за межі їхніх класів для спільного використання
- Великі групи об’єктів можна замінити невеликою кількістю об’єктів спільного використання
- Програма не залежить від ідентичності об'єктів

Зв’язок з іншими патернами

- **Компонувальник** часто поєднують з **Легковаговиком**, щоб реалізувати спільні гілки дерева та заощадити при цьому пам’ять
- Також часто **Легковаговик** використовують для об’єктів патернів **Стану** та **Стратегії**
- Патерн **Легковаговик** нагадує **Одинак**, але між ними є певні відмінності:
  - Об’єкт **Легковаговика** є незмінним
  - Може існувати безліч об’єктів **Легковаговиків** із різними станами