МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут **КНІТ**
Кафедра **ПЗ**

ЗВІТ

До лабораторної роботи № 5
З дисципліни: *"Моделювання та аналіз програмного забезпечення"*
На тему: *"Поведінкові шаблони"*

Лектор:
доц. каф. ПЗ
Сердюк П.В.

Виконав:
ст. гр. ПЗ-22
Чаус Олег

Прийняв:
викл. каф. ПЗ
Микуляк А. В.

« ____ » _____ 2023 р.

∑= ____ _____.

Львів – 2023

Тема роботи: Поведінкові шаблони.

Мета роботи: Здобути навички використання поведінкових шаблонів проектування при моделюванні програмних систем.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Шаблони поведінки (англ. behavioral patterns) — шаблони проектування, що пов'язані з алгоритмами та розподілом обов'язків поміж об'єктів. Мова в них йде не тільки про самі об'єкти та класи, але й про типові способи їхньої взаємодії. Шаблони поведінки характеризують складний потік керування, котрий досить важко прослідкувати під час виконання програми. Увага акцентована не на потоці керування, а на зв'язках між об'єктами. У шаблонах поведінки рівня класу використовується успадкування — щоб розподілити поведінку поміж різних класів. У шаблонах поведінки рівня об'єкта використовується композиція. Деякі з них описують, як за допомогою кооперації багато рівноправних об'єктів пораються із завданням, котре жодному з них поодинці не під силу. Тут важливо, як об'єкти отримують інформацію про існування один одного. Об'єкти-колеги можуть зберігати посилання один на одного, але це посилює ступінь зв'язаності системи. За максимального рівня зв'язаності кожному об'єкту довелось би мати інформацію про всі інші. Деякі з наведених шаблонів вирішують цю проблему.

ЗАВДАННЯ

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

Реалізувати 3 поведінкових шаблони на вибір з наступних:

- Ланцюжок відповідальностей (Chain of Responsibility),
- Стан (State),
- Стратегія (Strategy),
- Шаблонний метод (Template Method),
- Команда (Command),
- Посередник (Mediator),
- Спостерігач (Observer).

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

UML-ДІАГРАМА КЛАСІВ

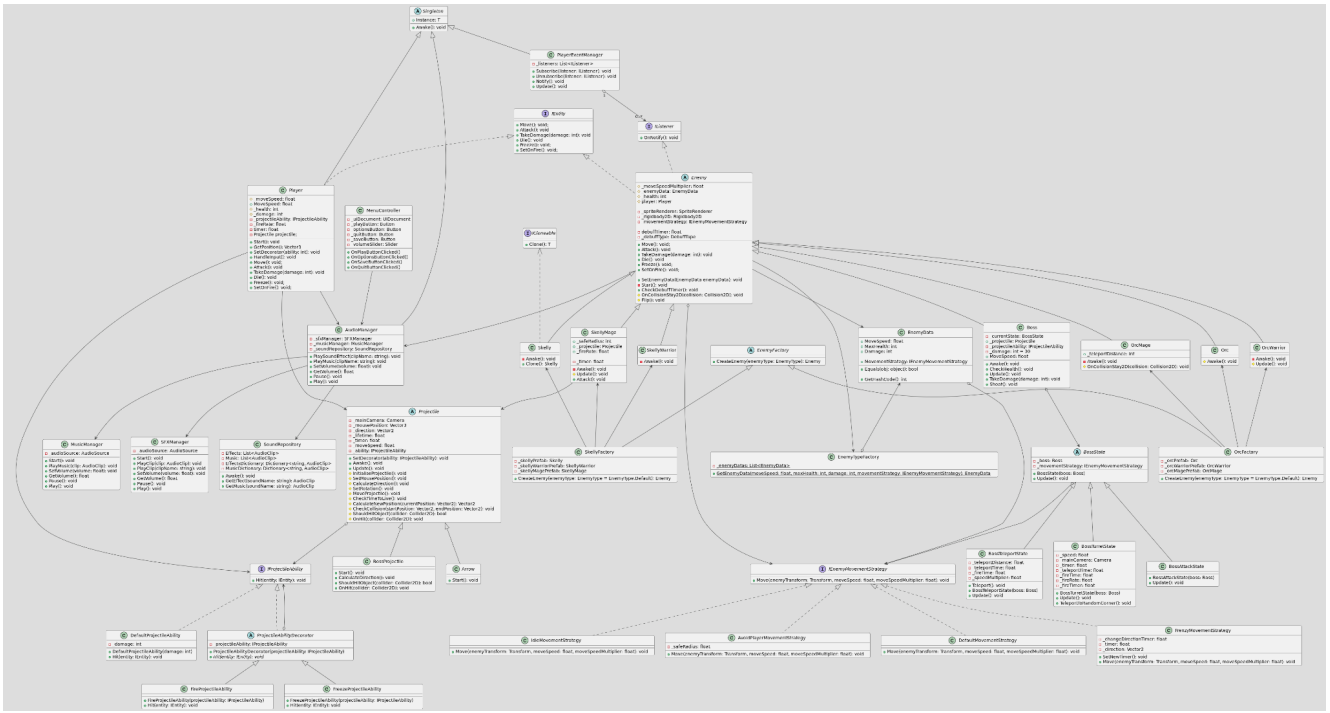

Рис. 1. Загальна UML-діаграма класів.

Було реалізовано три структурних патерни – Стан, Стратегія, Спостерігач.

- Стан

Стан — це поведінковий патерн проектування, що дає змогу об’єктам змінювати поведінку в залежності від їхнього стану. Ззовні створюється враження, ніби змінився клас об’єкта. Патерн Стан неможливо розглядати у відриві від концепції машини станів, також відомої як стейт-машина або скінченний автомат. Основна ідея в тому, що програма може знаходитися в одному з кількох станів, які весь час змінюють один одного. Набір цих станів, а також переходів між ними, визначений наперед та скінченний. Перебуваючи в різних станах, програма може по-різному реагувати на одні і ті самі події, що відбуваються з нею.

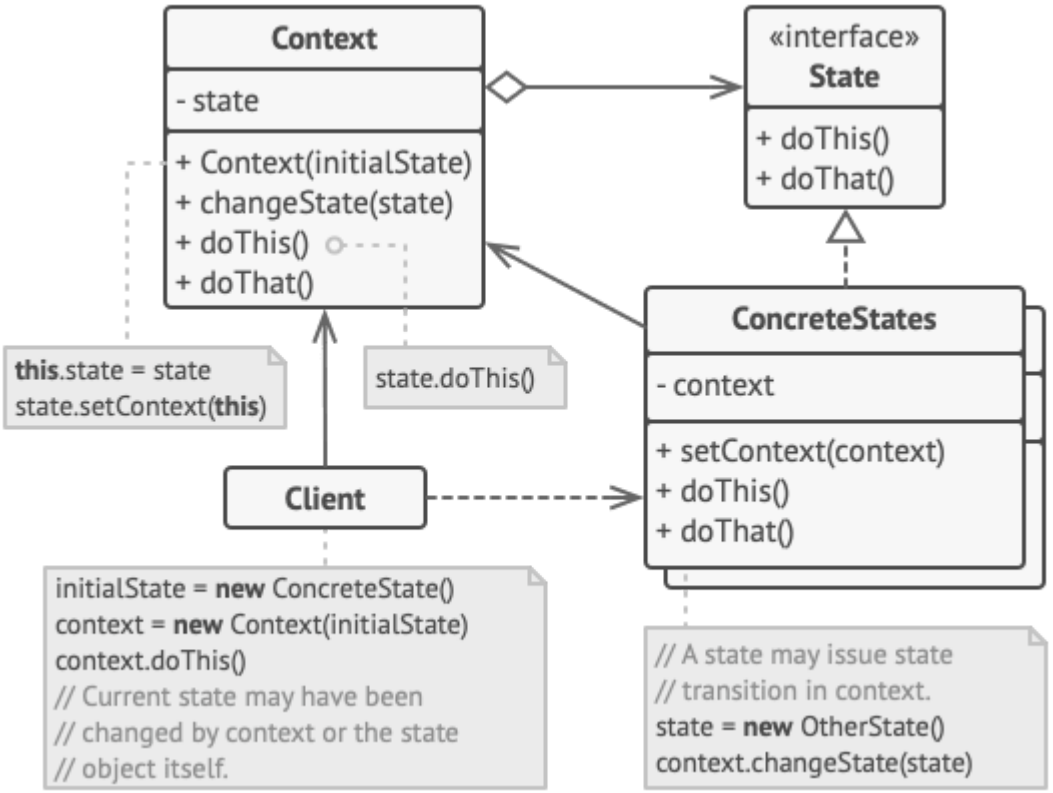

Рис. 2. Приклад використання State

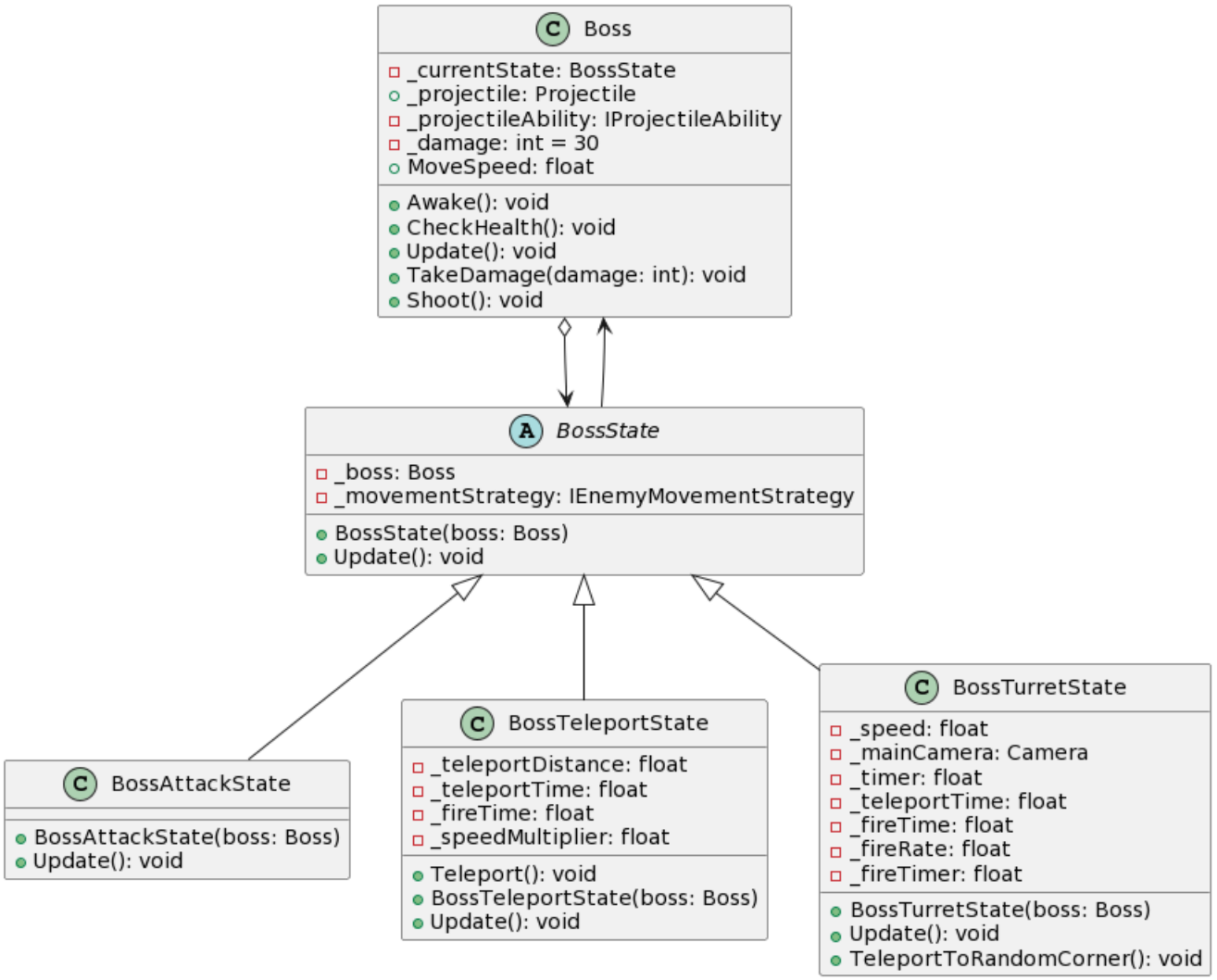

Рис. 3. Використання патерну State.

- Стратегія

Стратегія — це поведінковий патерн проектування, який визначає сімейство схожих алгоритмів і розміщує кожен з них у власному класі. Після цього алгоритми можна заміняти один на інший прямо під час виконання програми. Патерн Стратегія пропонує визначити сімейство схожих алгоритмів, які часто змінюються або розширюються, й винести їх до власних класів, які називають стратегіями. Замість того, щоб початковий клас сам виконував той чи інший алгоритм, він відіграватиме роль контексту, посилаючись на одну зі стратегій та делегуючи їй виконання роботи. Щоб змінити алгоритм, вам буде достатньо підставити в контекст інший об’єкт-стратегію. Важливо, щоб всі стратегії мали єдиний інтерфейс. Використовуючи цей інтерфейс, контекст буде незалежним від конкретних класів стратегій. З іншого боку, ви зможете змінювати та додавати нові види алгоритмів, не чіпаючи код контексту.

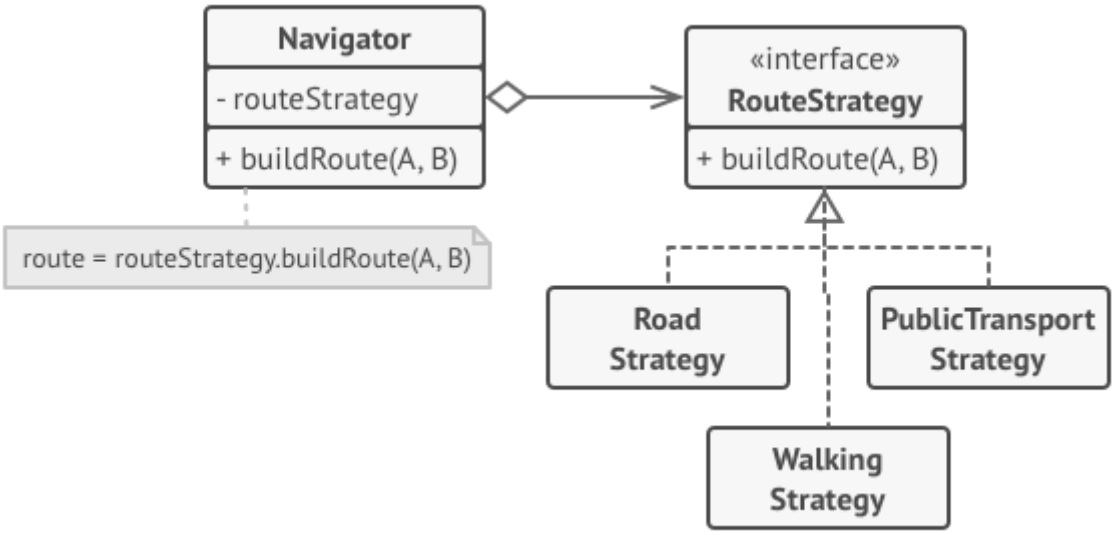

Рис. 4. Приклад використання Strategy.

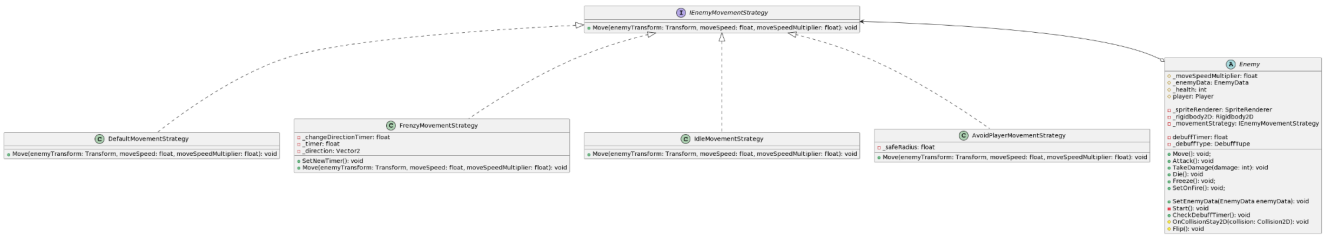

Рис. 5. Використання шаблону Strategy.

- Спостерігач

Спостерігач — це поведінковий патерн проектування, який створює механізм підписки, що дає змогу одним об’єктам стежити й реагувати на події, які відбуваються в інших об’єктах. Тобто шаблон Спостерігач визначає залежність типу «один до багатьох» між об'єктами таким чином, що при зміні стану одного об'єкта всіх залежних від нього сповіщають про цю подію. Шаблон «спостерігач» застосовується в тих випадках, коли система володіє такими властивостями:

● існує, як мінімум, один об'єкт, що розсилає повідомлення

● є не менше одного одержувача повідомлень, причому їхня кількість і склад можуть змінюватися під час роботи програми. Цей шаблон часто застосовують в ситуаціях, в яких відправника повідомлень не цікавить, що роблять одержувачі з наданою їм інформацією.

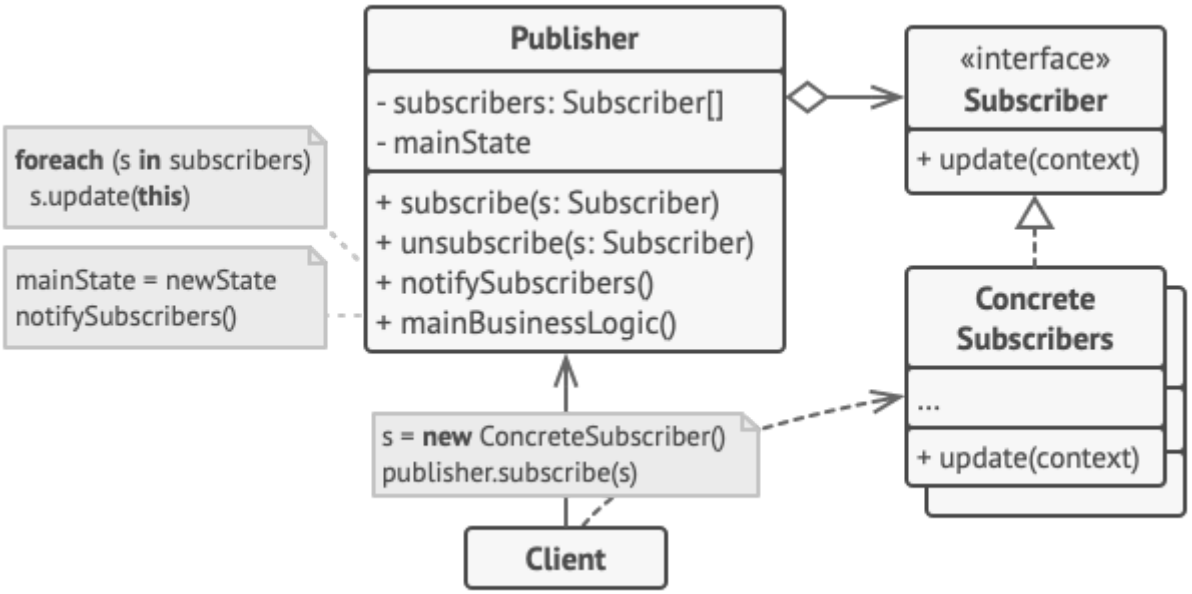

Рис. 6. Структура патерну Observer.

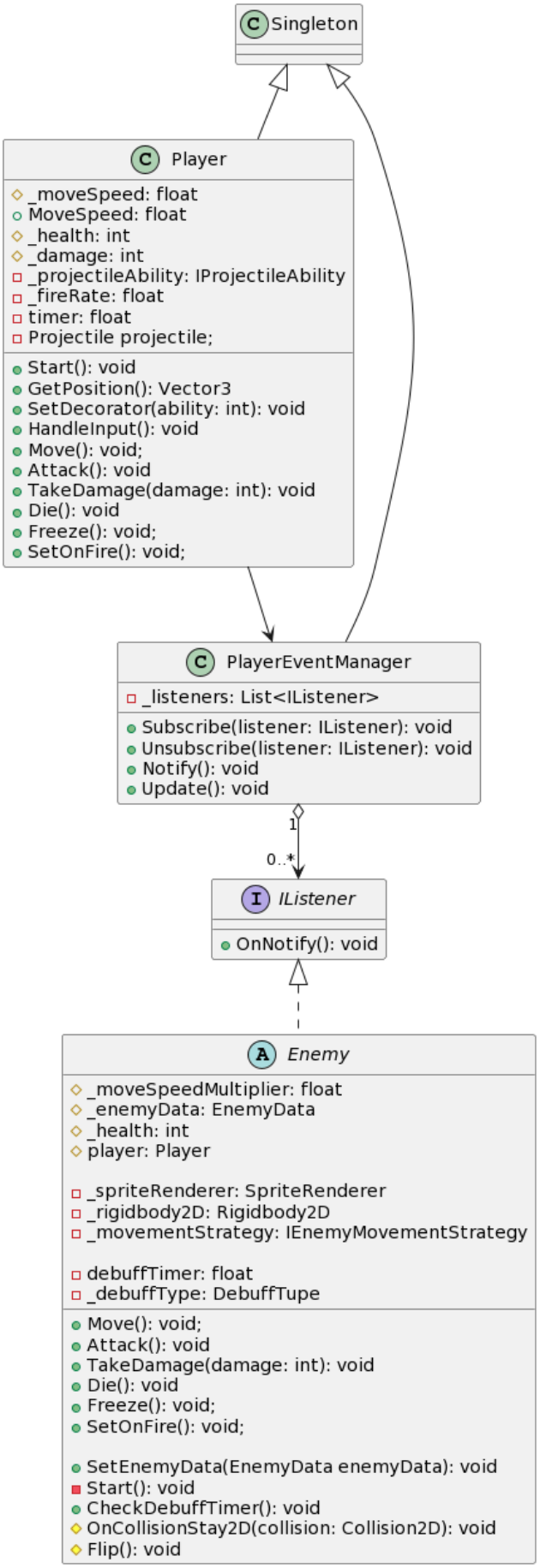
Singleton
Player
_moveSpeed: float
MoveSpeed: float
_health: int
_damage: int
_projectileAbility: IProjectileAbility
_fireRate: float
timer: float
Projectile projectile;
Start(): void
GetPosition(): Vector3
SetDecorator(ability: int): void
HandleInput(): void
Move(): void;
Attack(): void
TakeDamage(damage: int): void
Die(): void
Freeze(): void;
SetOnFire(): void;
PlayerEventManager
_listeners: List<IListener>
Subscribe(listener: IListener): void
Unsubscribe(listener: IListener): void
Notify(): void
Update(): void
1
0..*
IListener
OnNotify(): void
Enemy
_moveSpeedMultiplier: float
_enemyData: EnemyData
_health: int
player: Player
_spriteRenderer: SpriteRenderer
_rigidbody2D: Rigidbody2D
_movementStrategy: IEnemyMovementStrategy
debuffTimer: float
_debuffType: DebuffTupe
Move(): void;
Attack(): void
TakeDamage(damage: int): void
Die(): void
Freeze(): void;
SetOnFire(): void;
SetEnemyData(EnemyData enemyData): void
Start(): void
CheckDebuffTimer(): void
OnCollisionStay2D(collision: Collision2D): void
Flip(): void

Рис. 7. Використання Observer.

ВИСНОВКИ

Здобув навички використання поведінкових шаблонів проектування при моделюванні програмних систем.