МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

„КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**КУРСОВА РОБОТА**

з дисципліни «Програмування»   
на тему

**Шаблони проєктування в ООП. Комп'ютерна гра “WarMax”**

Виконав студент

ІІ курсу групи КП-02   
Слободзян Максим Вікторович залікова книжка КП-0217

Керівник роботи

доцент, к.т.н. Заболотня Т.М. Оцінка

(дата, підпис) 

КИЇВ 2022

# ЗМІСТ

[**ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ** 3](#_heading=h.2et92p0)

[**ВСТУП** 4](#_heading=h.tyjcwt)

[**1.**](#_heading=h.3dy6vkm) **СТРУКТУРНО-АЛГОРИТМІЧНА ОРГАНІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ КОМП'ЮТЕРНОЇ ГРИ “WARMAX”** 6

[1.1.](#_heading=h.1t3h5sf) Модульна організація програми 6

[1.2.](#_heading=h.4d34og8) Функціональні характеристики 7

[**2.**](#_heading=h.2s8eyo1) **ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ЗА ДОПОМОГОЮ ШАБЛОНІВ ПРОЄКТУВАННЯ** 8

[2.1.](#_heading=h.17dp8vu) Обґрунтування вибору та опис шаблонів проєктування для програмного забезпечення комп'ютерної гри “WarMax” 8

[2.2.](#_heading=h.3rdcrjn) Діаграма класів 19

[2.3.](#_heading=h.lnxbz9) Опис результатів роботи програми 20

[**ВИСНОВКИ** 26](#_heading=h.32hioqz)

[**СПИСОК** **ВИКОРИСТАНОЇ** **ЛІТЕРАТУРИ** 27](#_heading=h.1hmsyys)

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

**ПЗ** – (Програмне забезпечення) сукупність програм і програмних документів, необхідних для експлуатації цих програм.

**С#** – Об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET.

**DRY** – (від англ. Don`t Repeat Yourself) Принцип розробки програмного забезпечення, що направлений на уникнення дублювання інформації будь-якого вигляду.

**RPG** – (від англ. Role-Playing Game) Жанр відеоігор, де основна частина ігрового процесу полягає в управлінні персонажем чи групою персонажів, які досліджують ігровий світ, виконують різноманітні завдання (відомі як «квести», від англ. quest) та розвиваються, слідуючи сюжету.

**UML** – (від англ. Unified Modeling Language) Уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення.

# ВСТУП

Дана курсова робота присвячена розробці програмного забезпечення комп’ютерної гри “WarMax” за допомогою використання шаблонів проєктування. В наш час комп’ютерні ігри - невід’ємна частина дозвілля не тільки дітей, а й дорослих у всьому світі. Окрім розваг та веселих вражень, які приносить гра, відомо, що важливу роль у процесі формування особистості та розвитку мислення відіграє не тільки освітній фактор, але й характер ігрової діяльності. Різноманітні ігри допомагають ознайомитися ширше з навколишнім світом, розвивати логічне та креативне мислення, думати нестандартно, тому розроблене програмне забезпечення є досить актуальним. Дана тематика обрана для виконання курсової роботи тому, що результати абстрагування об’єктів у цій предметній галузі дозволяють застосувати вивчені принципи та методи об’єктно-орієнтованого програмування для створення програмного забезпечення, зокрема шаблони проєктування.

*Об’єктом* дослідження є процес гри в комп’ютерну гру “WarMax”.

*Метою роботи* є розроблення програмного забезпечення комп’ютерної гри “WarMax” з використанням шаблонів проєктування.

Для досягнення визначеної мети необхідно виконати такі *завдання*:

* абстрагувати об’єкти предметної галузі;
* розробити структурну організацію ПЗ за допомогою застосування основних принципів ООП та шаблонів проєктування;
* визначити та описати функціональні характеристики програми;
* обґрунтувати вибір шаблонів проєктування, використаних для побудови програми;
* розробити користувацький консольний інтерфейс;
* виконати реалізацію програмного забезпечення відповідно до вимог технічного завдання;
* виконати тестування розробленої програми;
* оформити документацію з курсової роботи.

Розроблене ПЗ комп’ютерної гри “WarMax” складається з чотирьох модулів: модуль взаємодії з користувачем, модуль придбання товарів, модуль ігрових квестів та модуль проведення битви.

Реалізовані шаблони проєктування: Абстрактна фабрика, Прототип, Замісник, Шаблонний метод, Стратегія, Спостерігач.

До функціональних можливостей програми належать: перевірки коректності вхідних даних, надання інформації щодо характеристик персонажа користувача та об’єктів ворогів, перегляд та купівля товарів у ігровому магазині, перегляд та статистика квестів, проведення битви тощо.

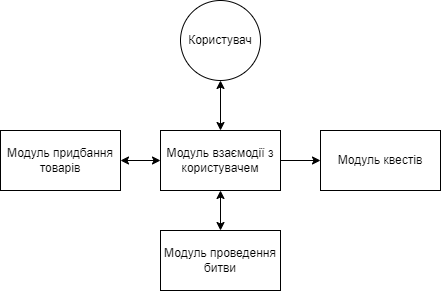
Для функціонування розробленої програми необхідно забезпечити наявність на комп’ютері 20 Мб вільного дискового простору та встановленого .Net Framework 6.0.

Розроблене програмне забезпечення може бути використане будь-ким, хто обожнює RPG-ігри та хоче розважитись й скоротати час.

Пояснювальна записка складається зі вступу, двох розділів, загальних висновків, списку використаних джерел (5-ти найменувань, з них 2 – іноземною мовою). Робота містить 20 рисунків. Загальний обсяг роботи – 27 друкованих сторінок, з них 24 сторінки основного тексту та 1 сторінка списку використаних джерел.

# СТРУКТУРНО-АЛГОРИТМІЧНА ОРГАНІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ КОМП'ЮТЕРНОЇ ГРИ “WARMAX”

## Модульна організація програми



*Рис. 1.1.1. Модульна організація програми*

У програмі можна виділити декілька основних логічних модулів, що мають певну самостійність і обмінюються один з одним даними. Взаємодія між модулями відбувається двома способами: за допомогою виклику методів та за допомогою надсилання подій.

**Модуль взаємодії з користувачем** виступає своєрідним мостом між користувачем та функціями системи за допомогою мінімалістичного консольного інтерфейсу - саме цей модуль отримує команду від користувача, перевіряє коректність вхідних даних та викликає методи інших модулів, а також отримує дані від інших модулів та виводить їх користувачу.

**Модуль придбання товарів** відповідає за внутрішньоігрові покупки користувача, а також перевіряє рівень доступності тих чи інших позицій асортименту в залежності від рівня та балансу персонажа користувача, наявності певних предметів тощо.

**Модуль квестів** відповідає за оновлення різноманітних завдань, видачу нагороди за їх успішне проходження та збереження статистики персонажа користувача.

**Модуль проведення битви** безпосередньо представляє собою поле битви. У модулі міститься об’єкт **персонажа** користувача та об’єкти **ворогів**, які взаємодіють між собою. В залежності від типу ворога, битва може мати цікаві та несподівані розв’язки. За допомогою надсилання подій модуль проведення битви може взаємодіяти з модулем ігрових квестів.

## Функціональні характеристики

Розроблене ПЗ дозволяє користувачу зручно керувати процесом гри за допомогою консольного інтерфейсу. При вході в гру користувач може обрати один з трьох класів ігрових персонажів. Після цього йому доступні такі можливості:

* перегляд даних про власного персонажа, його рівень, баланс, різноманітні ігрові характеристики та показники тощо;
* перегляд та придбання товарів у ігровому магазині;
* виконання різноманітних завдань, перегляд статистики їх проходження та досягнутих результатів;
* вибір локації проведення битви;
* проведення поєдинку з різними типами ворогів, які мають різноманітні характеристики та унікальні можливості.

Також користувач має змогу детальніше ознайомитися з ігровим процесом та особливостями різних компонентів гри за допомогою відповідної опції в користувацькому меню.

# ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ЗА ДОПОМОГОЮ ШАБЛОНІВ ПРОЄКТУВАННЯ

## Обґрунтування вибору та опис шаблонів проєктування для програмного забезпечення комп'ютерної гри “WarMax”

1. **Абстрактна фабрика**

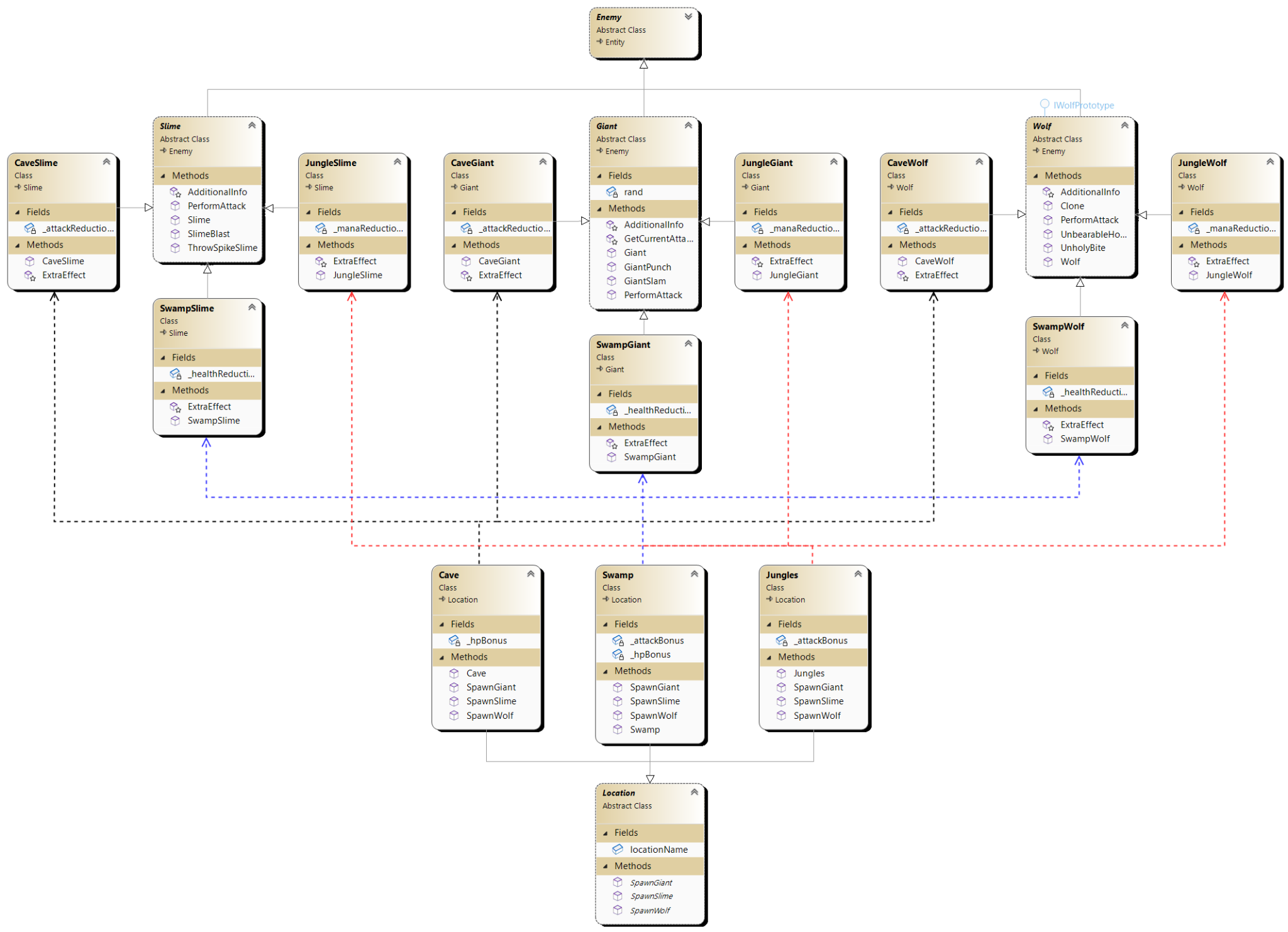
*Визначення:*

Абстрактна фабрика – породжувальний шаблон проєктування, що дає змогу створювати сімейства пов’язаних об’єктів, не прив’язуючись до конкретних класів створюваних об’єктів. UML діаграма класів представлена на рис. 2.1.1.

*Обґрунтування використання шаблону:*

Даний шаблон доцільно використати при побудові програми через те, що заздалегідь невідомо яку локацію для проходження захоче обрати користувач для генерації ворогів. Він дозволяє клієнту не залежати від того який і яким саме чином буде створений ворог, адже він очікує загальний інтерфейс ворога та працює із загальним інтерфейсом фабрики.

Використання такого шаблону значно спрощує програму. Він звільняє клієнтський код від прив’язки до конкретних класів продукту: для того, щоб створювати ворогів з різних локацій, не потрібно вносити зміни в код клієнта, достатньо просто вибрати для використання іншу конкретну реалізацію загального інтерфейсу. Також абстрактна фабрика виділяє код виробництва продуктів в одне місце, спрощуючи підтримку коду.



*Рис. 2.1.1. UML діаграма класів, які входять до шаблону “Абстрактна фабрика”*

*Учасники шаблону:*

* **AbstractFactory** (Location) – оголошує інтерфейс для операцій, що створюють абстрактні об’єкти-вороги;
* **ConcreteFactory** (Swamp, Cave, Jungles) – реалізує методи абстрактної фабрики, що створюють абстрактні об’єкти-вороги;
* **AbstractProduct** (Wolf, Slime, Giant) – оголошує інтерфейс об’єкта-ворога;
* **ConcreteProduct** (JungleWolf, JungleSlime, JungleGiant, CaveWolf, CaveSlime, CaveGiant, SwampWolf, SwampSlime, SwampGiant) – визначає об’єкт-ворог, що створюється відповідною конкретною фабрикою.

1. **Прототип**

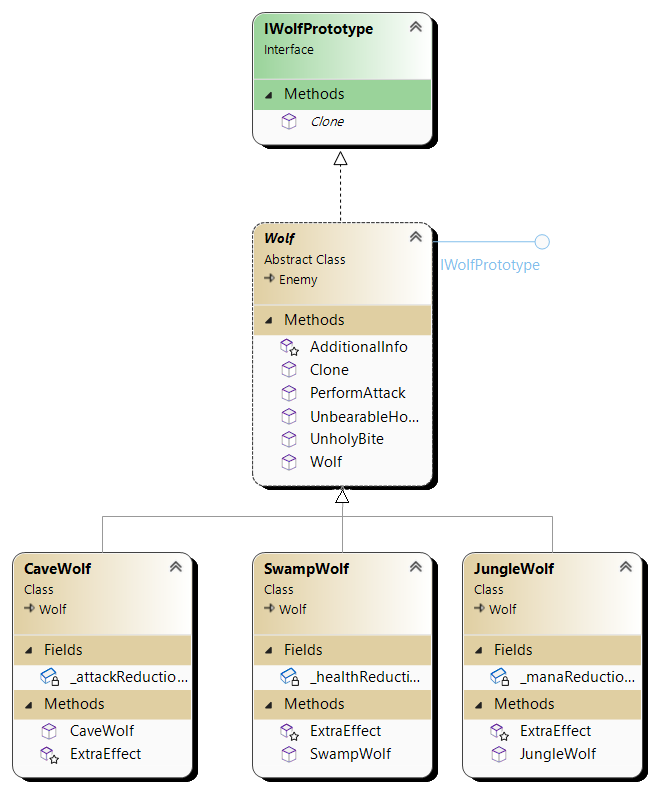
*Визначення:*

Прототип – породжувальний шаблон проєктування, що дає змогу копіювати об’єкти, не вдаючись у подробиці їхньої реалізації. UML діаграма класів представлена на рис. 2.1.2.

*Обґрунтування використання шаблону:*

Під час розробки програми виникає завдання – скопіювати об’єкт певного ворога. Можна створити порожній об’єкт такого ж класу і по черзі копіювати значення усіх полів, але частина його стану може бути приватною. До того ж, код, що копіює, стане залежним від класів об’єктів, які він копіює.

Шаблон прототип доручає процес копіювання самим об’єктам, які треба скопіювати. Він вводить загальний інтерфейс для всіх об’єктів, що підтримують клонування. Це дозволяє копіювати об’єкти, не прив’язуючись до їхніх конкретних класів.



*Рис. 2.1.2. UML діаграма класів, які входять до шаблону “Прототип”*

*Учасники шаблону:*

* **Prototype** (IWolfPrototype) – оголошує інтерфейс для операції клонування – єдиний метод *clone*;
* **ConcretePrototype** (CaveWolf, JungleWolf, SwampWolf) – реалізує операцію клонування самого себе.

1. **Замісник**

*Визначення:*

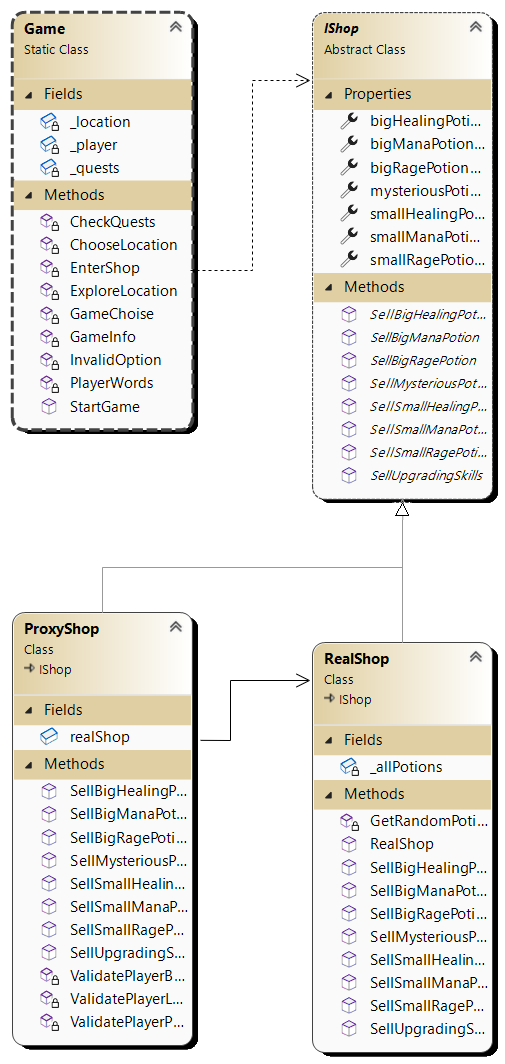
Замісник – структурний шаблон проєктування, що дає змогу підставляти замість реальних об’єктів спеціальні об’єкти-замінники. Ці об’єкти перехоплюють виклики до оригінального об’єкта, дозволяючи зробити щось до чи після передачі виклику оригіналові. UML діаграма класів представлена на рис. 2.1.3.

*Обґрунтування використання шаблону:*

У програмі існує об’єкт магазину, з яким може взаємодіяти користувач. Проте ми повинні контролювати доступ до об’єкту, оскільки деякі позиції вимагають наявність необхідного рівня та певного балансу, при цьому не змінюючи поведінку клієнта.

Шаблон замісник пропонує створити новий клас-дублер, який має той самий інтерфейс, що й оригінальний службовий клас. При отриманні запиту від клієнта об’єкт-замісник перевіряє певні умови, після чого за необхідності може викликати потрібний метод службового об’єкту.

Такий об’єкт-дублер можна підставляти всюди, де очікується оригінальний об’єкт. Це дозволяє контролювати сервісний об’єкт непомітно для клієнта.

**

*Рис. 2.1.3. UML діаграма класів, які входять до шаблону “Замісник”*

*Учасники шаблону:*

* **Client** (Game) – працює з об’єктами через інтерфейс магазину. При цьому об’єкт магазину, з яким він працює, підмінюється об’єктом-замісником;
* **Subject** (IShop) – оголошує загальний інтерфейс операцій для магазину та його замісника;
* **RealSubject** (RealShop) – реалізує інтерфейс операцій магазину;
* **Proxy** (ProxyShop) – об’єкт-замісник, що зберігає посилання на оригінальний об’єкт магазину та контролює до нього доступ.

1. **Шаблонний метод**

*Визначення:*

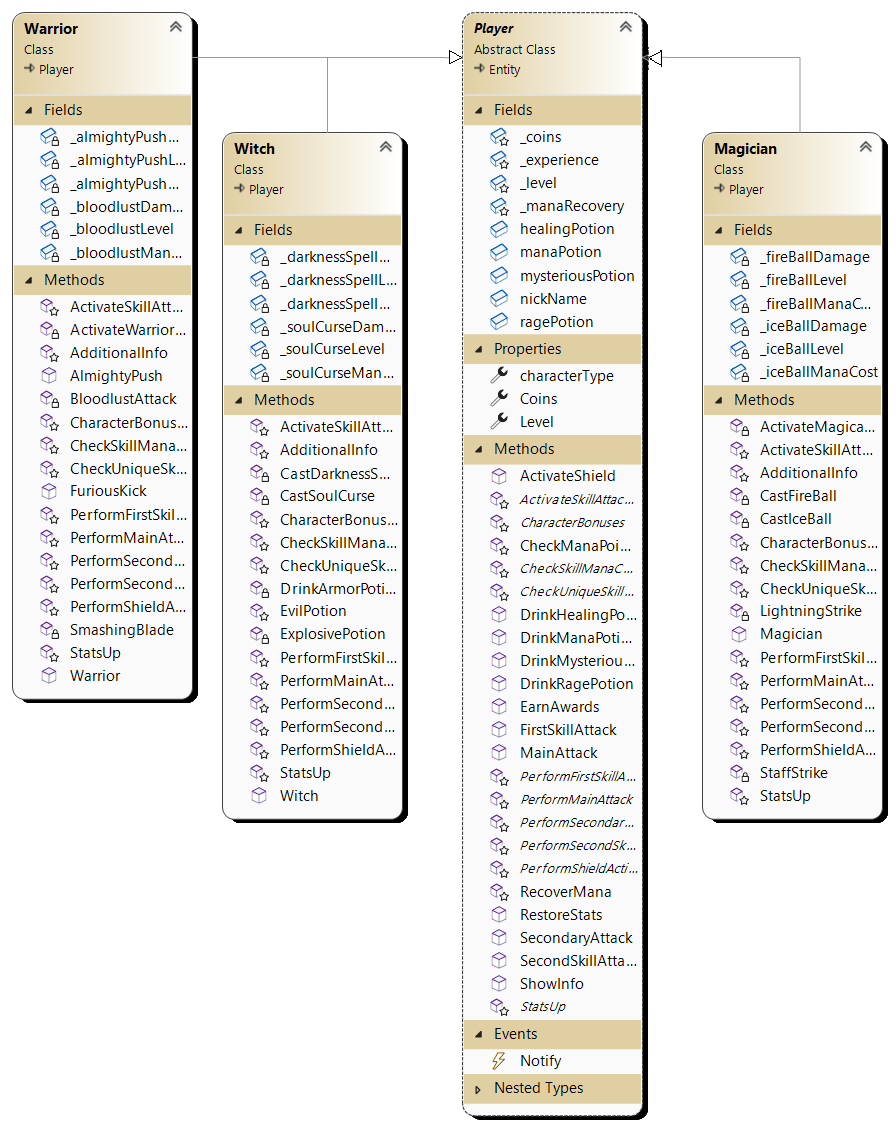
Шаблонний метод – поведінковий шаблон проєктування, який визначає кістяк алгоритму, перекладаючи відповідальність за деякі його кроки на підкласи. Патерн дозволяє підкласам перевизначати кроки алгоритму, не змінюючи його загальної структури. UML діаграма класів представлена на рис. 2.1.4.

*Обґрунтування використання шаблону:*

Даний шаблон доцільно використати для різних класів персонажа користувача. Хоч і унікальні можливості, типи атак, відновлення ресурсу та бонуси за певні дії різні та відрізняються між собою, проте код класів персонажів містить досить багато спільного в частині загальних дій при атаці, здобуттю нагород тощо. Наприклад, кожен персонаж отримує монети та досвід за успішне закінчення битви, при цьому виконується перевірка чи не досяг персонаж нового рівня, у разі чого потрібно підвищити загальні характеристики та перевірити чи не можна покращити рівень унікальних можливостей. Або ж процес загальної атаки: спочатку треба виконати певну атаку, потім в залежності від типу персонажа отримати бонуси до статистики та відновити ресурс. І таких методів є кілька. Дублювання такої кількості коду жорстко порушить принцип DRY.

До того ж інший код, який працює з об’єктами цих класів, наповнений умовами, що перевіряють тип обробника перед початком роботи. Весь цей код можна спростити, якщо злити всі класи в одне ціле або звести їх до загального інтерфейсу.

Шаблонний метод пропонує розбити алгоритми на послідовність кроків, описати ці кроки в окремих методах і викликати їх в шаблонних методах один за одним. Це дозволить підкласам перевизначити деякі кроки алгоритмів, залишаючи без змін їх структуру та інші кроки, які для цього підкласу не є важливими. Також цей шаблон унеможливлює дублювання коду в кількох класах, які відрізняються деталями, але мають схожу поведінку.



*Рис. 2.1.4. UML діаграма класів, які входять до шаблону “Шаблонний метод”*

*Учасники шаблону:*

* **AbstractClass** (Player) – визначає кроки певних алгоритмів та містить шаблонні методи, які складаються з викликів цих кроків;
* **ConcreteClass** (Warrior, Magician, Witch) – перевизначають деякі кроки алгоритму, проте аж ніяк не самі шаблонні методи.

1. **Стратегія**

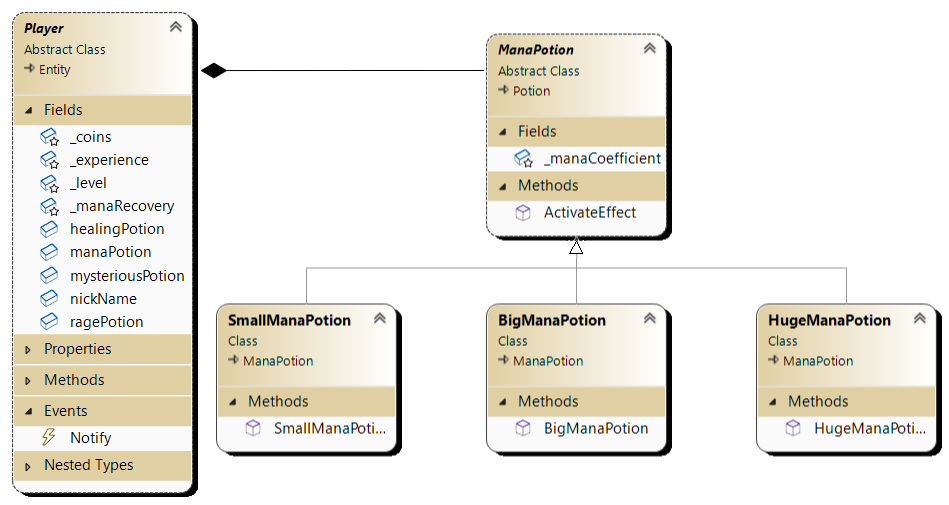
*Визначення:*

Стратегія – поведінковий шаблон проєктування, 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. UML діаграма класів представлена на рис. 2.1.5.

*Обґрунтування використання шаблону:*

Даний шаблон доцільно використати через наявність великого сімейства схожих алгоритмів, які впродовж програми будуть часто змінюватись. Їх можна винести до власних класів, які називають стратегіями.

Гравець може відновлювати здоров’я, енергію, збільшувати значення атаки. Оскільки всі алгоритми відрізняються лише показниками, що покращуються або погіршуються, та величиною нового значення, то доцільно розробити класи стратегій – зілля. Тепер замість того, щоб початковий клас сам виконував той чи інший алгоритм, він відіграє роль контексту, посилаючись на певну стратегію та делегує їй виконання роботи. Щоб змінити алгоритм, достатньо підставити в контекст інший об’єкт стратегію. За рахунок спільного інтерфейсу контекст буде незалежним від конкретних класів стратегій. Також можна легко змінювати та додавати нові види алгоритмів, не чіпаючи при цьому код контексту.



*Рис. 2.1.5. UML діаграма класів, які входять до шаблону “Стратегія”*

На рисунку зображено одну з реалізацій шаблону Стратегія, всі інші реалізації є схожими та мають однакову структурну будову.

*Учасники шаблону:*

* **Context** (Player) – зберігає посилання на об’єкт стратегії, працює з ним через загальний інтерфейс стратегій;
* **Strategy** (ManaPotion) – визначає спільний для всіх варіацій алгоритму інтерфейс;
* **ConcreteStrategy** (SmallManaPotion, BigManaPotion, HugeManaPotion) – реалізує певну варіацію алгоритму.

1. **Спостерігач**

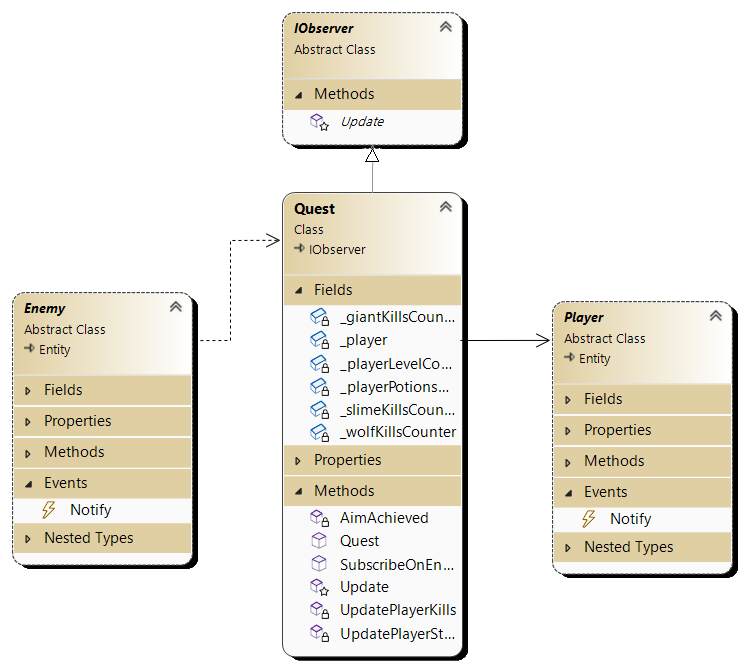
*Визначення:*

Спостерігач – поведінковий шаблон проєктування, який створює механізм підписки, що дає змогу одним об’єктам стежити й реагувати на події, які відбуваються в інших об’єктах. UML діаграма класів представлена на рис. 2.1.6.

*Обґрунтування використання шаблону:*

Даний шаблон доцільно використати через те, що у програмі є об’єкти, які змінюють свій стан (персонаж користувача, вороги), та об’єкт, які зацікавлені у зміні цього стану (об’єкт ігрових квестів). Об’єкт квестів може кожен раз перевіряти зміну на той чи інший стан інших об’єктів, але код стане занадто громіздким у такому випадку, адже ці перевірки потрібно виконувати у різних місцях програми. До того ж, немає потреби сильно зв’язувати об’єкти, які взаємодіють; об’єктам, що змінюють свій стан, не важливо кого вони повинні сповіщати про зміни.

Шаблон Спостерігач дозволяє розробити систему сповіщення об’єктами своїх “підписників” про зміни, що відбуваються з ними у процесі їх діяльності. Об’єкти персонажа та користувача перетворюються на своєрідних “видавців” які сповіщають про зміну свого стану, а об’єкт квестів – “підписник” – має змогу “підписатися” на ці сповіщення. Це не зв’язує об’єкти напряму і видавцям байдуже кого сповіщати.

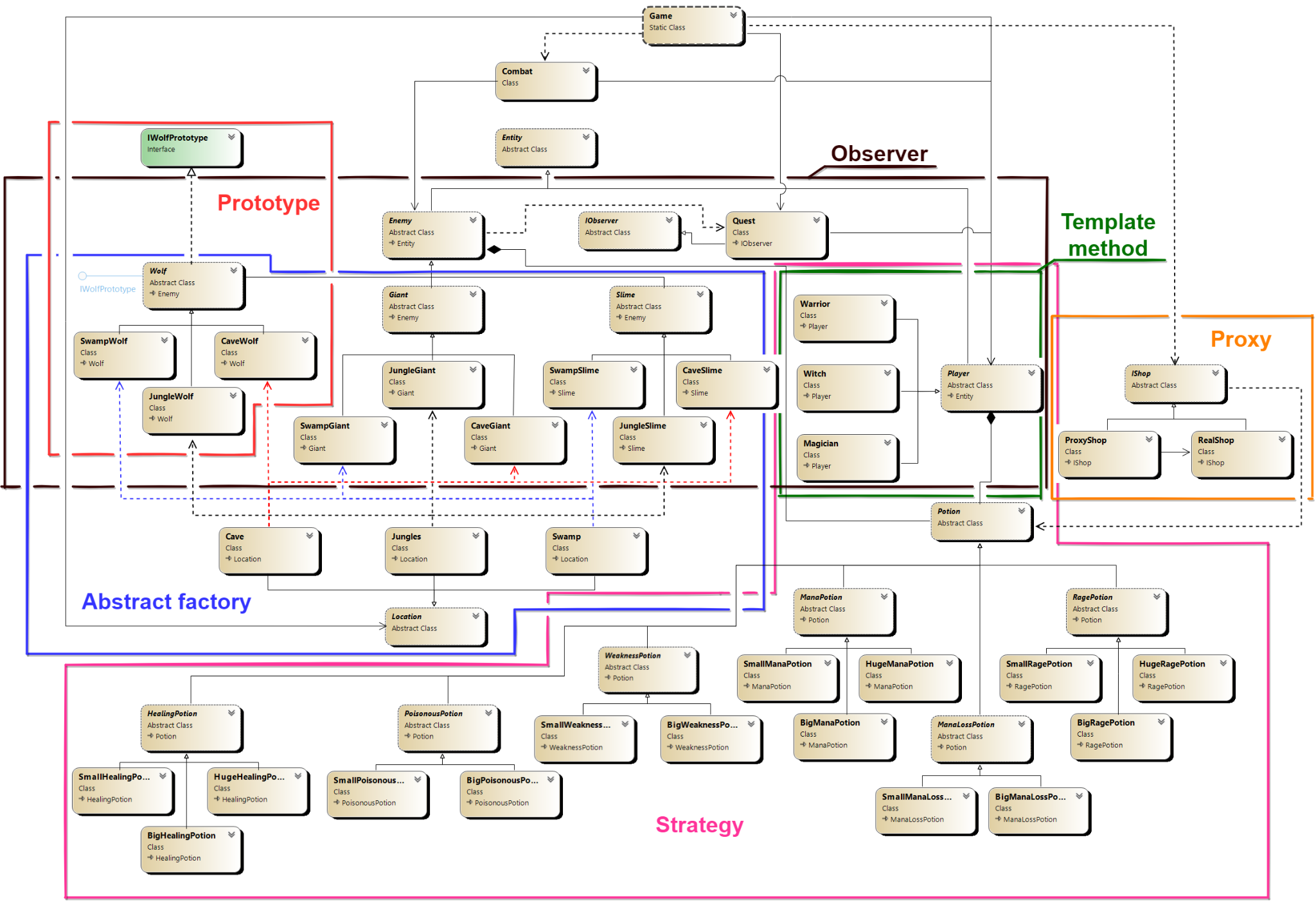


*Рис. 2.1.6. UML діаграма класів, які входять до шаблону “Спостерігач”*

*Учасники шаблону:*

* **Publisher/Subject** (Enemy, Player) – володіє внутрішнім станом, зміни якого цікавлять підписника;
* **Subscriber/Observer** (IObserver) – визначає інтерфейс, який повинен реалізувати конкретний підписник;
* **Concrete Subscriber/Observer** (Quests) – виконує певні функції у відповідь на сповіщення, які надходять від видавців.

## Діаграма класів



*Рис. 2.2 UML діаграма класів комп’ютерної гри “WarMax”*

## 

## Опис результатів роботи програми

При початку роботи програми користувачу пропонується ввести особистий ігровий нікнейм та обрати тип персонажа, якого він бажає:

## 

*Рис. 2.3.1. Ілюстрація введення даних персонажу гри*

Після коректно введених даних користувач потрапляє до головного меню програми:

## 

*Рис. 2.3.2. Ілюстрація головного меню гри*

## 

Користувач має змогу переглянути характеристики персонажа, унікальні можливості, баланс, рівень, інвентар тощо, виберемо для цього необхідний пункт меню:

## 

*Рис. 2.3.3. Інформація про персонажа користувача*

## 

Користувач має змогу переглянути та придбати різні товари, що продаються у ігровому магазині:

## 

*Рис. 2.3.4. Ілюстрація асортименту магазину*

## 

## 

При спробі покупки товару персонажем, який володіє недостатнім рівнем/балансом, або якщо такий предмет вже наявний у персонажа в інвентарі, програма сповістить користувача про це:

## 

*Рис. 2.3.5. Відповідь програми на невдале придбання товару*

## 

Користувач має змогу переглянути поточні квести та статистику їх виконання, перейшовши у відповідне меню програми:

## 

*Рис. 2.3.6. Ілюстрація квестів персонажа гри*

## 

Для того, аби розпочати дослідження світу, спочатку потрібно обрати локацію, куди повинен відправитись на пошуки пригод персонаж користувача:

## 

*Рис. 2.3.7. Вибір ігрової локації користувачем*

## 

Після успішного вибору локації персонаж йде на пошуки пригод, де зустрічається з одним із видів ворогів, які обираються та генеруються псевдовипадково. Починається подія битви:

## 

*Рис. 2.3.8. Ілюстрація початку події битви*

## 

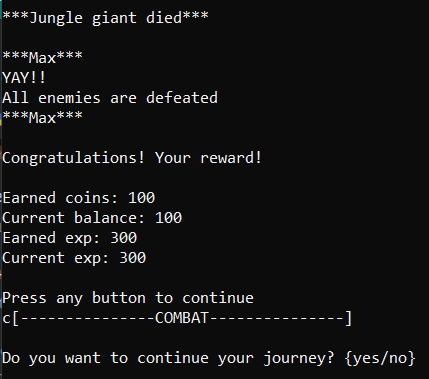
У ході битви персонаж та ворог по черзі виконують комбінації атак. Комбінації атак ворога визначаються псевдовипадковим чином, для проведення атак або перегляду інформації про супротивника користувач має спеціальне мінімалістичне меню:

## 

*Рис. 2.3.9. Меню гравця під час битви*

## 

Після закінчення битви користувач отримує повідомлення про перемогу та зарахування бонусів або поразку. Також користувачу пропонується продовжити подорож:



*Рис. 2.3.10. Ілюстрація закінчення битви*

## 

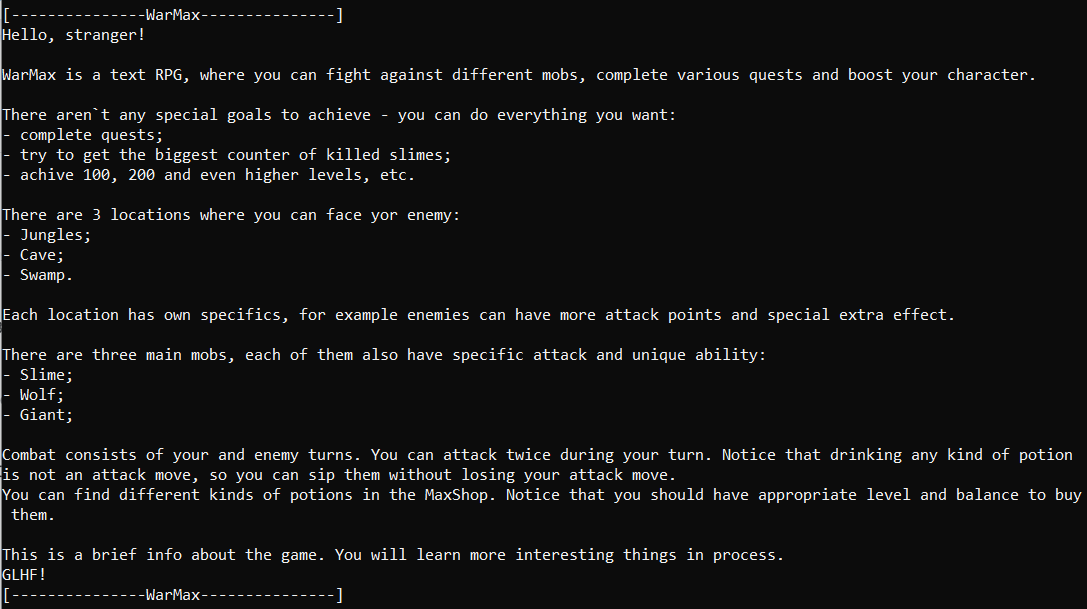
При некоректно введених даних чи виборі неіснуючої опції програма повідомляє користувача про невірну дію:

## 

*Рис. 2.3.11. Ілюстрація обробки невірно введеної команди*

## 

Детальніше з особливостями гри користувач може ознайомитись за допомогою відповідного пункту меню:



*Рис. 2.3.12. Ілюстрація ознайомлення користувача з ігровим процесом*

# ВИСНОВКИ

Метою даної курсової роботи було розроблення ПЗ гри “WarMax” з використанням шаблонів проєктування. Підставою для розроблення стало завдання на виконання курсової роботи з дисципліни «Програмування» студентами ІІ курсу кафедри ПЗКС НТУУ «КПІ ім. І.Сікорського».

Для досягнення поставленої мети у повному обсязі виконано завдання, визначені у аркуші завдання на курсову роботу; розроблено графічні матеріали; реалізовано всі вимоги до програмного продукту, програмного та апаратного забезпечення, наведені у технічному завданні; створено відповідну документацію.

Розроблене програмне забезпечення представляє собою розважальну комп'ютерну RPG-гру “WarMax”, яка дозволяє користувачу змагатися проти різноманітних ворогів, виконувати різноманітні квести та купувати різні товари в ігровому магазині.

Програму створено на основі використання таких структурних, породжувальних та поведінкових шаблонів проєктування, як: Абстрактна фабрика, Прототип, Замісник, Шаблонний метод, Спостерігач, Стратегія.

Для розроблення програмного забезпечення використано мову програмування C# та платформу .NET версії 6.0.

Перспективним напрямом подальшого дослідження даної тематики є розширення функціональності даної гри, впровадження різних механік поєдинку, розроблення нових типів ворогів та персонажів гри.

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. C# 3.0 Design Patterns / Judith Bishop – 2007. – 22 c., – 101 c., – 122 c., – 139 c., –158 c., –210 c.
2. Design Patterns via C#. Прийоми об'єктно-орієнтованого проєктування/ Олександр Шевчук, Дмитро Охрименко, Андрій Касьянов – 2015. – 35 c., – 72 c., – 155 c., – 228 c., – 265 c., – 270 c.
3. Шаблоны проектирования. Новый подход к объектно-ориентированному анализу и проектированию / Алан Шаллоуей, Джеймс Р. Тротт. — М. : «Вильямс», 2002. — 288 с.
4. Refactoring Guru [Електронний ресурс]: [Веб-сайт]. - Режим доступу: https://refactoring.guru/uk/design-patterns/ (дата звернення 27.06.2022) - Назва з екрана
5. Software Design Patterns [Електронний ресурс]: [Веб-сайт]. - Режим доступу: https://www.geeksforgeeks.org/software-design-patterns (дата звернення 27.06.2022) - Назва з екрана