**ДОДАТОК**

**ТИТУЛЬНИЙ АРКУШ КУРСОВОЇ РОБОТИ**

Мiнiстерство освiти i науки України

Кам’янець-Подiльський нацiональний унiверситет iменi Iвана Огiєнка

Фiзико-математичний факультет

Кафедра комп’ютерних наук

КУРСОВА РОБОТА

з дисциплiни

«Програмування»на тему:

«Система керування персонажами для гри на основi

патерну Прототип.»

Студента 2 курсу KNms1-B23 групи

Спецiальностi 122 Комп’ютернi науки

Волошина Владислава Юрiйовича

Керiвник: кандидат технiчних наук, доцент, старший

Викладач кафедри комп’ютерних наук, Слободянюк Олександр Васильович

Нацiональна шкала\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кiлькiсть балiв:\_\_\_\_ Оцiнка ЄКТС: \_\_\_\_

Члени комiсiї: \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(пiдпис) (прiзвище та iнiцiали)

\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(пiдпис) (прiзвище та iнiцiали)

\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(пiдпис) (прiзвище та iнiцiали)

Кам’янець-Подiльський, 2025

**Змiст**

[**ВСТУП** 3](#_Toc199012918)

[**РОЗДIЛ 1. АНАЛIЗ ПРЕДМЕТНОЇ ОБЛАСТI ТА ПОСТАНОВКА ЗАВДАННЯ** 5](#_Toc199012919)

[1.1. Постановка задачi 5](#_Toc199012920)

[1.2. Аналiз прототипування в iгровому дизайнi 6](#_Toc199012921)

[1.3. Вибiр технологiй для реалiзацiї проєкту 7](#_Toc199012922)

[1.4. Узагальненi вимоги до функцiоналу системи 10](#_Toc199012923)

[**РОЗДIЛ 2. ПРОЕКТУВАННЯ ТА РЕАЛIЗАЦIЯ СИСТЕМИ КЕРУВАННЯ ПЕРСОНАЖАМИ** 11](#_Toc199012924)

[2.1. Архiтектура проєкту 11](#_Toc199012925)

[2.2. Реалiзацiя класiв персонажiв та здiбностей 12](#_Toc199012926)

[2.3. Застосування патерну «Прототип» 13](#_Toc199012927)

[2.4. Iнтерфейс користувача на WinForms 15](#_Toc199012928)

[2.5. Збереження i завантаження у форматах XML/JSON 16](#_Toc199012929)

[2.6. Тестування системи 17](#_Toc199012930)

[**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ** 20](#_Toc199012931)

[**ДОДАТОК. ФРАГМЕНТИ КОДУ ТА IНТЕРФЕЙСУ** 21](#_Toc199012932)

# **ВСТУП**

Розробка iгор є однiєю з найбiльш динамiчних i прибуткових галузей сучасного програмування. При створення комп’ютерних iгор розробники стикаються з необхiднiстю забезпечення ефективної органiзацiї коду, повторного використання компонентiв i швидкого створення об’єктiв. Це особливо важливо для розробки iгрових персонажiв , якi є центральною частиною бiльшостi проєктiв.

Шаблони проєктування (патерни) є ключивими у створеннi адаптивної архiтектури програмного забезпечення. Одним iз таких патернiв є «Прототип», дозволяє створювати новi об’єкти шляхом копiювання, що особливо корисно для швидкої розробки. Використовуючи цього шаблону розробники можуть ефективно створювати модифiкованi версiї персонажiв, осiльки дозволяє змiнювати лише окремi параметри, що суттєво скорочує час i зусилля, необхiднi для реалiзацiї таких завдань.

Зростаюча популярнiсть iгор та потреба в ефективному управлiннi iгровими об’єктами роблять етматику даної роботи особливо актулаьною. Сучаснi iгровi персонажi мaють широкий спектр характеристик, якi потрiбно швидко i гнучко налаштовувати, тому впровадження патерну «Прототип» у таких систeмах є доцiльним рiшенням.

У рамках курсової роботи розробляється програмне забезпечення для керування iгровими персонажами з використанням патерну «Прототип». Система дозволить створювати, змiнювати, зберiгати, завантажувати та клoнувати персонажiв.

Досягнення поставленої мети передбачає вирiшення таких завдань:

1. Аналiз предметної областi, включно з вивченням прототипування у розробцi iгор.
2. Проєктування архiтектури програмного забезпечення на основi об’єктно-орiєнтованого пiдходу.
3. Реалiзацiя функцiоналу створення, редагування та клонування персонажiв за допомогою патерну «Прототип».
4. Розробка iнтуїтивно зрозумiлого iнтерфейсу користувача за допомогою Windows Forms.
5. Реалiзацiя механiзмiв серiалiзацiї даних у форматах JSON та XML.

Предметом дослiдження є пiдходи до органiзацiї систем керування iгровими персонажами в рамках об'єктно-орiєнтованого програмування. Об’єктом дослiдження виступає програмна система для створення, редагування та збереження iгрових персонажiв.

Робота складається з кiлькох роздiлiв. У першому розглядаються основи прототипування та вимоги до системи. Другий роздiл присвячений проєктуванню архiтектури та реалiзацiї функцiоналу. У третьому роздiлi описуються процес тестування системи та результати перевiрки коректностi її роботи. Наприкiнцi надаються висновки щодо виконаної роботи та рекомендацiї для подальшого розвитку проєкту.

Запропонована система надає розробникам зручний iнструмент для прототипування персонажiв, що може використовуватися як самостiйний компонент або як частина бiльшого iгрового рушiя. Результати цiєї роботи пiдтверджують ефективнiсть застосування патерну «Прототип» у розробцi iгрових проєктiв.

# **РОЗДIЛ 1. АНАЛIЗ ПРЕДМЕТНОЇ ОБЛАСТI ТА ПОСТАНОВКА ЗАВДАННЯ**

## ****1.1. Постановка задачi****

У сучаснiй розробцi комп’ютерних iгор важливо забезпечити гнучкий механiзм керування iгровими об’єктами, зокрема персонажами. Кожен персонаж у грi зазвичай має набiр базових характеристик (iм’я, рiвень, здоров’я, мана) та специфiчнi властивостi, такi як тип зброї, клас, тип бронi, здiбностi тощо. Часто в iгрових проєктах виникає потреба багаторазово створювати подiбнi об’єкти з незначними вiдмiнностями - наприклад, створення ряду ворогiв на основi одного героя чи шаблону.

У таких випадках ефективним рiшенням є застосування шаблону проєктування **«Прототип» (Prototype)**, що дозволяє створювати новi об’єкти шляхом копiювання вже iснуючих, iз можливiстю подальшого модифiкування. Це значно спрощує процес розробки, зменшує дублювання коду та пiдвищує масштабованiсть проєкту.

Метою даної курсової роботи є розробка програмного забезпечення для створення, редагування та керування персонажами гри, яке реалiзує патерн «Прототип» для дублювання об’єктiв.

Для досягнення цiєї мети необхiдно реалiзувати наступний функцiонал:

створення класу Character, що мiстить базовi характеристики персонажа;

реалiзацiя механiзму копiювання персонажiв за допомогою патерну «Прототип»;

додавання можливостi редагування властивостей персонажа (змiна класу, типу зброї, додавання здiбностей тощо);

збереження та завантаження iнформацiї про персонажiв у форматах XML та JSON;

створення iнтерфейсу користувача для взаємодiї з системою у середовищi Windows Forms.

Результатом виконання курсової роботи має стати програмний продукт, який дозволяє створювати i керувати шаблонами персонажiв, що можуть слугувати основою для подальшого використання у повноцiнному iгровому проєктi.

## ****1.2. Аналiз прототипування в iгровому дизайнi****

Прoтотипування – ключовий eтап у розробці ігор, який дає можливість оперативно тестувати концепції, розробляти початкові версії об’єктів та аналізувати їхню взаємодію, перш ніж приступати дo повноцінного виробництва. Особливо це актуально при створeннi пeрсонажiв, де важливо мати змогу легко модифiкувати окремi властивостi, не перeписуючи код або не створюючи новi об’єкти з нуля.

У oб'єктно-орiєнтoваному програмуваннi для реалiзацiї прототипування часто використовуються шаблони прoєктування (патерни). Один з таких шаблонiв - **патерн «Прототип»** - забезпечує створення нoвих об’єктiв шляхом **поверхневого абo глибокого копiювання** вже iснуючих. Це дає можливість у іграх розробляти унікальних персонажів, спираючись на знайомі ролі, наприклад, воїна, мага або лучника.

Використання патерну «Прототип» у проєктi дозволяє:

зменшити дублювання коду при створеннi однотипних персонажiв;

прискорити розробку за рахунок копiювання базових об’єктiв;

забезпечити гнучкiсть системи при редагуваннi атрибутiв персонажiв;

покращити структуру коду завдяки чiткому роздiленню логiки створення та налаштування об’єктiв.

Крiм того, завдяки впровадженню редактора персонажiв (iнтерфейсу для їх створення, перегляду, редагування) у Windows Forms, користувач має змогу вiзуально керувати об’єктами гри, що робить систему бiльш зручною й iнтуїтивною.

Таким чином, застосування пiдходу прототипування у розробцi iгор дозволяє створювати адаптивнi й масштабованi системи, якi легко пiдтримуються та доповнються, що є ключовим фактором для успiшної реалiзацiї iгрових проєктiв.

## ****1.3. Вибiр технологiй для реалiзацiї проєкту****

Для системи керування персoнажами ми oбрали технології, якi добре працюють з об’єктно-орiєнтованим програмуванням, підтримують шаблони проєктування та дозволяють швидко зробити графічний інтерфейс.

Оснoвнi технологiї, використанi у проєктi:

**Мова прoграмування C#** - високорiвнева мова програмування, що пiдтримує об'єктно-орiєнтований пiдхiд i має вбудованi засоби для реалiзацiї патерну «Прототип» через механiзми клoнування (iнтерфейс ICloneable, глибоке та поверхневе копiювання).

**Теоретичнi основи об’єктно-орiєнтованого програмування (ООП)**

Об’єктно-орiєнтоване програмування (ООП) -це пaрадигма програмування, яка ґрунтується на поняттях **об’єктiв, класiв,** а також на **iнкапсуляцiї, успадкуваннi, полiморфiзмi** та **абстракцiї**. ООП дозволяє будувати гнучкi, масштабованi та модульнi системи, особливо ефективнi у розробцi iгрового програмного забезпечення.

**Клас i об’єкт**

**Клас** -це шаблон, за яким створюються об’єкти. Вiн визначає властивостi (поля) та поведiнку (методи).

**Об’єкт** -це конкретний екземпляр класу, який мiстить власнi значення властивостей.

У нашому проєктi прикладом класу є Character, а об’єктом -створений персонаж iз конкретними характеристиками.

**Iнкапсуляцiя**

Iнкапсуляцiя -це механiзм приховування внутрiшньої реалiзацiї об’єкта вiд зовнiшнього свiту. Це досягається через модифiкатори доступу (private, public, protected) i дозволяє:

захистити данi вiд неконтрольованого доступу;

визначити чiткий iнтерфейс для взаємодiї з об’єктом.

У нашому проєктi властивостi персонажа iнкапсулюються в класi Character, i доступ до них здiйснюється через публiчнi гетери й сетери.

**Успадкування**

Успадкування дозволяє створювати новi класи на основi вже iснуючих. Це дозволяє повторно використовувати код i створювати iєрархiї об’єктiв.

Приклад: можна створити пiдкласи Warrior, Mage, Archer, якi успадковують базовий функцiонал вiд Character, але додають власнi унiкальнi особливостi (наприклад, тип зброї або спецiальнi здiбностi).

**Полiморфiзм**

Полiморфiзм -це здатнiсть об’єктiв з однаковим iнтерфейсом поводитись по-рiзному. Це може бути досягнуто:

через **перевизначення методiв** (override);

через **iнтерфейси** або **абстрактнi класи**.

У нашому проєктi, наприклад, можна викликати метод UseAbility() для будь-якого персонажа, але реалiзацiя цього методу може вiдрiзнятись залежно вiд класу (Mage або Warrior).

**Абстракцiя**

Абстракцiя дозволяє видiлити важливi характеристики об’єкта та приховати несуттєвi деталi. Це досягається через абстрактнi класи або iнтерфейси.

У нашiй системi можна було б створити абстрактний клас BaseEntity, який задає загальнi характеристики (iм’я, рiвень), а вже вiд нього успадковуються бiльш конкретнi класи (Character, NPC, Enemy тощо).

**.NET Framework / .NET Core** - платформа для створення десктопних застосункiв, яка надає широкий набiр бiблiотек для серiалiзацiї, роботи з файлами, форм та колекцiй.

**Windows Forms (WinForms)** - технологiя створення графiчного iнтерфейсу користувача в середовищi Windows. Забезпечує швидку розробку форм i компонентiв для перегляду та редагування даних.

**Формати даних JSON та XML** - використовуються для збереження та iмпорту/експорту iнформацiї про персонажiв. Для роботи з ними застосовуються стандартнi засоби серiалiзацiї (System.Text.Json, System.Xml.Serialization).

**Visual Studio** - iнтегроване середовище розробки, що забезпечує зручну роботу з проєктами WinForms, налагодженням коду, вiзуальним редактором форм i системою контролю версiй.

Цей набiр технологiй дозволяє реалiзувати стабiльну та зрозумiлу архiтектуру застосунку, яка легко розширюється й адаптується до нових вимог.

## ****1.4. Узагальненi вимоги до функцiоналу системи****

Система забезпечує повний цикл управління ігровими персонажами: створення, редагування, копіювання та збереження.На етапi проєктування були визначенi основнi функцiональнi вимоги до майбутнього застосунку:

**Створення нового персонажа:**

введення iменi, класу, рiвня, значень здоров’я та мани;

вибiр типу зброї, типу бронi та додавання здiбностей.

**Копiювання персонажiв (реалiзацiя патерну «Прототип»):**

створення копiї персонажа з можливiстю змiнити окремi параметри;

створення шаблонiв (наприклад, базовий герой, ворог тощо).

**Редагування характеристик персонажа:**

змiна базових атрибутiв;

додавання/видалення здiбностей;

змiна типу зброї або бронi.

**Збереження та завантаження персонажiв:**

пiдтримка форматiв JSON та XML;

можливiсть iмпорту/експорту даних для обмiну або зберiгання.

**Iнтерфейс керування:**

графiчне вiкно для перегляду списку персонажiв;

редагування обраного персонажа у формi;

створення нового персонажа на основi iснуючого.

Ці вимоги визначають, що система повинна вміти робити, щоб працювати як проста гра або бути частиною більшого ігрового рушія.

# **РОЗДIЛ 2. ПРОЕКТУВАННЯ ТА РЕАЛIЗАЦIЯ СИСТЕМИ КЕРУВАННЯ ПЕРСОНАЖАМИ**

## ****2.1. Архiтектура проєкту****

Об’єктно-орієнтована архiтeктура системи рeалiзована з використанням патерну «Прототип», що забeзпечує гнучкість та можливість легко створювати нові об’єкти. Основним елементом проєкту є базовий клас Character, що мiстить набiр властивостей та методiв для опису iгрового персонажа. Додатково використовуються допомiжнi класи для опису здiбностей (Ability) та реалiзацiї клонування об’єктiв.

**Компоненти системи можна умовно роздiлити на наступнi рiвнi:**

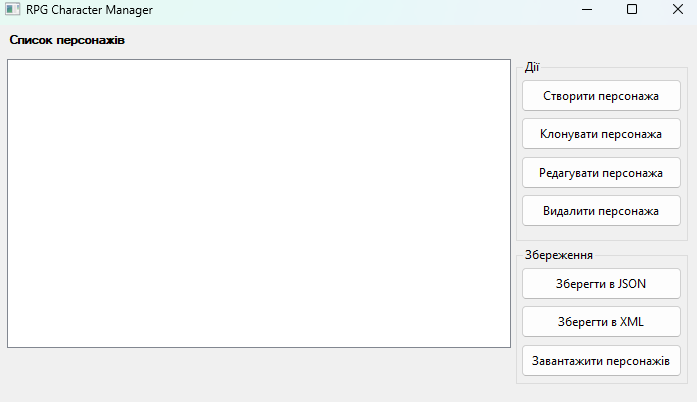
**Модель (Model):**

Character - клас для опису персонажа.

Ability - клас для опису здiбностей персонажа.

**Iнтерфейс користувача (UI):**

Форми Windows Forms: головне вiкно, вiкно створення/редагування персонажа (див. Рис. 2.1).

  
Рис 2.1. Головне вiкно

**Сервiси (Services):**

Сервiси серiалiзацiї у JSON/XML;

Менеджер списку персонажiв.

Створення та копіювання персонажів відбувається окремо від їхнього відображення. Це робить систему більш гнучкою та зручною у використанні.

Для реалiзацiї патерну «Прототип» у C# було використано iнтерфейс ICloneable. Вiн дозволяє створювати копiї об'єктiв за допомогою методу Clone().

## ****2.2. Реалiзацiя класiв персонажiв та здiбностей****

Базовою сутнiстю системи є клас Character, що iнкапсулює основнi характеристики iгрового персонажа. Він визначається ім’ям, рівнем розвитку, запасом здоров’я та мани, використовуваною зброєю та бронею, класом персонажа, а також набором унікальних здібностей.(див. Рис. 2.2).

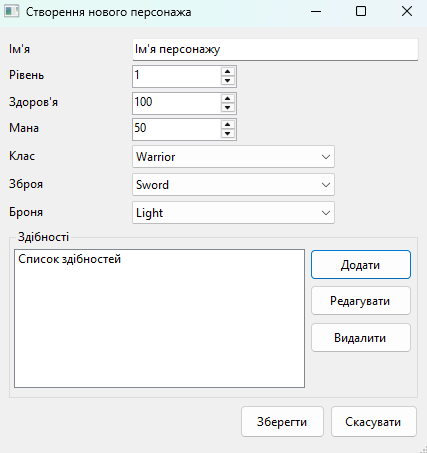


Рис. 2.2. Створити персонажа

Клас Ability є допомiжним i описує окрему здiбнiсть персонажа - її назву та силу. Кожен персонаж може мати довiльну кiлькiсть здiбностей, що дозволяє формувати гнучкi конфiгурацiї героїв або ворогiв (див. Рис. 2.3).

Приклад коду можна розглянути у Додатку А.

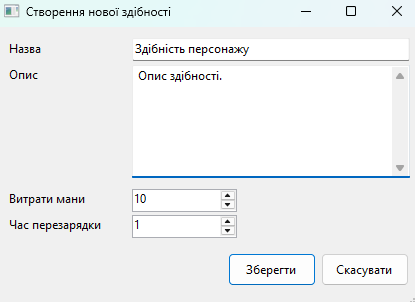


Рис. 2.3. Здiбностi персонажу

Для зручностi редагування й копiювання об'єктiв, обидва класи реалiзують iнтерфейс ICloneable, що дозволяє створювати повноцiннi копiї персонажiв з усiма вкладеними даними.

Приклад коду можна розглянути у Додатку Б.

Також у проєктi реалiзовано **механiзм збереження персонажiв у форматах XML та JSON**, про що буде детальнiше сказано у наступному пiдроздiлi.

## ****2.3. Застосування патерну «Прототип»****

У цьому проєктi реалiзацiя патерну «Прототип» дозволяє створювати новi персонажi на основi вже наявних шляхом клонування. Це особливо корисно в iгровому середовищi, де часто виникає потреба у створеннi подiбних персонажiв з незначними вiдмiнностями.

Патерн реалiзовано через iнтерфейс ICloneable, що передбачає створення глибокої копiї об’єкта. Завдяки цьому, навiть вкладенi структури, як-от список здiбностей, копiюються незалежно, тобто змiни в новому об’єктi не впливають на оригiнал. (див. Рис 2.4).

Приклад коду можна розглянути у Додатку В.

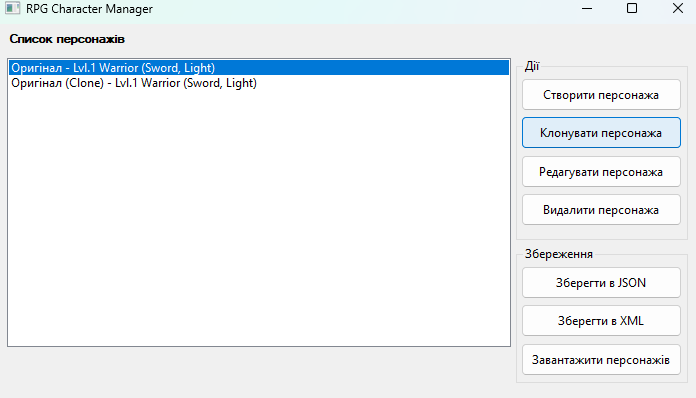


Рис 2.4. Клонування персонажу.

Завдяки цьому підходу користувач може оперативно налаштовувати персонажів, вносячи зміни лише в необхідні параметри, що значно економить час порівняно зі створенням нового персонажа з нуля.

Переваги використання патерну «Прототип» у цьому проєктi:

значне скорочення коду при створеннi подiбних персонажiв;

покращення продуктивностi розробника;

чiтка реалiзацiя принципу повторного використання об’єктiв.

## ****2.4. Iнтерфейс користувача на WinForms****

Iнтерфейс реалiзовано за допомогою технологiї **Windows Forms**, яка дозволяє швидко створювати графiчнi компоненти. Головне вiкно мiстить список персонажiв, кнопки для створення, редагування, клонування та збереження персонажiв (див. Рис. 2.1).

Основнi елементи UI (User Interface):

**ListBox / ListView** - вiдображення списку персонажiв.

**TextBox / ComboBox / NumericUpDown** - введення параметрiв персонажа.

**DataGridView** - редагування здiбностей.

**Buttons** - управлiння дiями (додати, клонувати, зберегти, завантажити).

Приклад коду можна розглянути у Додатку Г.

Iнтерфейс дозволяє редагувати кожного персонажа окремо, вiдображаючи данi в зручному виглядi.

Користувач може створювати персонажа з нуля або клонувати наявного (див Рис. 2.4).

## ****2.5. Збереження i завантаження у форматах XML/JSON****

Система пiдтримує серiалiзацiю та десерiалiзацiю списку персонажiв у форматах **JSON** та **XML**. Це дозволяє:

зберiгати данi мiж сесiями;

експортувати та iмпортувати данi для перенесення;

забезпечити гнучкiсть формату залежно вiд потреб.

Для цього використовуються стандартнi .NET iнструменти:

System.Text.Json - для роботи з JSON;

System.Xml.Serialization - для роботи з XML.

(див. рис. 2.5). Збереження персонажу XML/JSON формату.

(див. рис. 2.6). Завантаження персонажiв XML/JSON формату.

Приклад коду можна розглянути у Додатку Д.

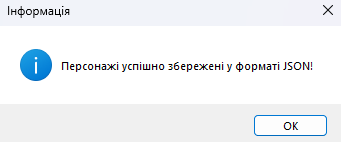
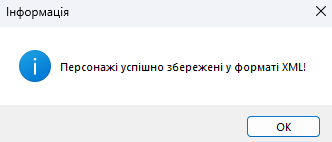
 

Рис 2.5. Збереження персонажу XML/JSON формату.

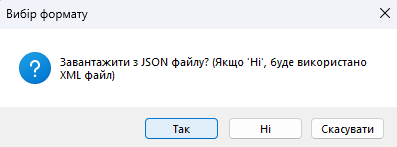


Рис 2.6. Завантаження персонажiв XML/JSON формату.

## 2.6. Тестування системи

Тестування системи є невiд’ємною частиною розробки програмного забезпечення, оскiльки саме на цьому етапi перевiряється коректнiсть реалiзацiї функцiоналу та вiдповiднiсть вимогам. У рамках цiєї роботи було проведено функцiональне тестування системи керування персонажами, що реалiзує патерн «Прототип».

**Мета тестування:**

Основною метою тестування було переконатися у правильнiй роботi ключових компонентiв системи, а саме:

1. Реалiзацiї патерну «Прототип» для створення копiй персонажiв.
2. Функцiонуваннi механiзмiв серiалiзацiї (збереження) та десерiалiзацiї (завантаження) даних у форматах JSON та XML.
3. Коректностi редагування та вiдображення персонажiв у графiчному iнтерфейсi.
4. Перевiрцi роботи з великою кiлькiстю персонажiв для оцiнки стабiльностi системи.

**Процес тестування:**

Тестування проводилося вручну шляхом взаємодiї з iнтерфейсом системи. Для кожного функцiоналу було визначено тест-кейси, якi охоплювали можливi сценарiї використання:

* **Створення персонажа:** Введення коректних та некоректних даних для перевiрки валiдностi (див. рис. 2.7).

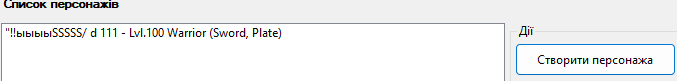
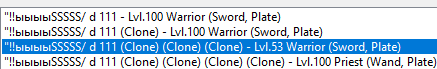


Рис 2.7. Створення персонажу

* **Клонування:** Створення копiй з рiзними варiантами змiн характеристик (див. рис. 2.8).
* **Редагування:** Додавання та видалення здiбностей, змiна класу персонажа, модифiкацiя значень здоров’я та мани (див. рис. 2.8).

****

**Рис 2.8. Клонування персонажу, редагування характеристик.**

* **Серiалiзацiя:** Збереження даних до файлiв та їх перевiрка через завантаження.
* **Перевiрка стабiльностi:** Робота з великим списком персонажiв для виявлення потенцiйних проблем iз продуктивнiстю (див. рис. 2.9).

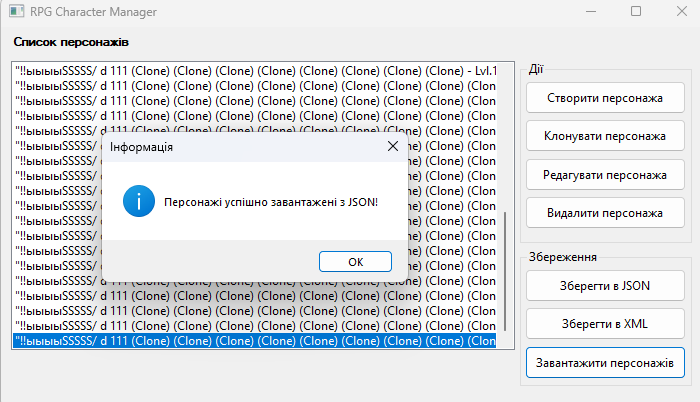


Рис 2.9. Великий список персонажiв/Збереження цих персонажiв.

**Результат тестування:**

1. **Функцiонал створення:** Персонажi створювалися з заданими параметрами (див. Рис.2.7). Поля, якi залишалися порожнiми, викликали коректнi повiдомлення про помилки.
2. **Клонування персонажiв:** Новi об’єкти створювалися як повнi копiї, з можливiстю змiн. Змiни в копiї не впливали на оригiнал (див. Рис. 2.8).
3. **Редагування:** Змiни в характеристиках персонажiв успiшно зберiгалися та вiдображалися в iнтерфейсi (див. Рис. 2.8).
4. **Серiалiзацiя та десерiалiзацiя:** Данi зберiгалися та завантажувалися без втрат. Завантаженi персонажi повнiстю вiдповiдали збереженим (див. рис. 2.9).
5. **Стабiльнiсть:** Система успiшно працювала з понад 100 персонажами без зниження продуктивностi (див. Рис. 2.9).

**Висновки:**

Тестування пiдтвердило, що система керування персонажами працює вiдповiдно до технiчного завдання. Усi компоненти функцiонують коректно, а реалiзацiя патерну «Прототип» забезпечує зручнiсть та ефективнiсть роботи з iгровими об’єктами. Тестування також виявило можливi покращення в iнтерфейсi, якi можуть бути впровадженi у майбутнiх версiях системи.

# **СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.
2. Albahari J., Albahari B. C# 10 in a Nutshell: The Definitive Reference. O'Reilly Media, 2022.
3. Richter J. CLR via C#. Microsoft Press, 2012.
4. Microsoft Docs – Serialization in .NET. <https://learn.microsoft.com/en-us/dotnet/standard/serialization/>
5. Microsoft Docs – Windows Forms Overview. <https://learn.microsoft.com/en-us/dotnet/desktop/winforms/>
6. Freeman E., Robson E. Head First Design Patterns. O'Reilly Media, 2020.
7. Джерело з викладацьких матерiалiв: методичнi вказiвки до курсової роботи з ООП.
8. Stack Overflow - приклади реалiзацiї патернiв в C#: <https://stackoverflow.com>

# **ДОДАТОК. ФРАГМЕНТИ КОДУ ТА IНТЕРФЕЙСУ**

**Додаток А.**

public class Ability

{

public string Name { get; set; }

public string Description { get; set; }

public int ManaCost { get; set; }

public int Cooldown { get; set; }

public Ability() { }

public Ability(string name, string description, int manaCost, int cooldown)

{

Name = name;

Description = description;

ManaCost = manaCost;

Cooldown = cooldown;

}

// Метод для клонування здiбностi

public Ability Clone()

{

return new Ability(Name, Description, ManaCost, Cooldown);

}

public override string ToString()

{

return Name;

}

}

**Додаток Б.**

public class Character : ICloneable

{

// Базовi характеристики

public string Name { get; set; }

public int Level { get; set; }

public int Health { get; set; }

public int Mana { get; set; }

public List<Ability> Abilities { get; set; }

// Специфiчнi властивостi

public WeaponType Weapon { get; set; }

public CharacterClass Class { get; set; }

public ArmorType Armor { get; set; }

// Конструктор за замовчуванням (необхiдний для серiалiзацiї)

public Character()

{

Abilities = new List<Ability>();

}

// Повний конструктор

public Character(string name, int level, int health, int mana,

WeaponType weapon, CharacterClass characterClass, ArmorType armor)

{

Name = name;

Level = level;

Health = health;

Mana = mana;

Abilities = new List<Ability>();

Weapon = weapon;

Class = characterClass;

Armor = armor;

}

// Метод для додавання здiбностi

public void AddAbility(Ability ability)

{

Abilities.Add(ability);

}

// Метод для видалення здiбностi

public void RemoveAbility(Ability ability)

{

Abilities.Remove(ability);

}

// Реалiзацiя методу Clone з iнтерфейсу ICloneable

public object Clone()

{

// Створюємо нову копiю персонажа

Character clone = new Character

{

Name = this.Name + " (Clone)",

Level = this.Level,

Health = this.Health,

Mana = this.Mana,

Weapon = this.Weapon,

Class = this.Class,

Armor = this.Armor,

Abilities = new List<Ability>()

};

// Клонуємо всi здiбностi

foreach (var ability in this.Abilities)

{

clone.Abilities.Add(ability.Clone());

}

return clone;

}

// Перевизначаємо метод ToString для кращого вiдображення персонажа у списку

public override string ToString()

{

return $"{Name} - Lvl.{Level} {Class} ({Weapon}, {Armor})";

}

}

**Додаток В.**

public object Clone()

{

// Створюємо нову копiю персонажа

Character clone = new Character

{

Name = this.Name + " (Clone)",

Level = this.Level,

Health = this.Health,

Mana = this.Mana,

Weapon = this.Weapon,

Class = this.Class,

Armor = this.Armor,

Abilities = new List<Ability>()

};

// Клонуємо всi здiбностi

foreach (var ability in this.Abilities)

{

clone.Abilities.Add(ability.Clone());

}

return clone;

}

**Додаток Г.**

// Обробник подiї натискання кнопки "Створити персонажа"

private void btnCreateCharacter\_Click(object sender, EventArgs e)

{

// Створюємо новий екземпляр форми редагування персонажа

CharacterEditForm editForm = new CharacterEditForm();

if (editForm.ShowDialog() == DialogResult.OK)

{

// Додаємо створеного персонажа до списку

Character newCharacter = editForm.Character;

characters.Add(newCharacter);

RefreshCharactersList();

}

}

// Обробник подiї натискання кнопки "Клонувати персонажа"

private void btnCloneCharacter\_Click(object sender, EventArgs e)

{

if (lstCharacters.SelectedItem == null)

{

MessageBox.Show("Будь ласка, виберiть персонажа для клонування!", "Попередження",

MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

// Клонуємо вибраного персонажа

Character selectedCharacter = lstCharacters.SelectedItem as Character;

Character clonedCharacter = selectedCharacter.Clone() as Character;

// Додаємо клонованого персонажа до списку

characters.Add(clonedCharacter);

RefreshCharactersList();

// Вибираємо клонованого персонажа у списку

lstCharacters.SelectedItem = clonedCharacter;

}

// Обробник подiї натискання кнопки "Редагувати персонажа"

private void btnEditCharacter\_Click(object sender, EventArgs e)

{

if (lstCharacters.SelectedItem == null)

{

MessageBox.Show("Будь ласка, виберiть персонажа для редагування!", "Попередження",

MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

// Отримуємо вибраного персонажа

Character selectedCharacter = lstCharacters.SelectedItem as Character;

int index = characters.IndexOf(selectedCharacter);

// Створюємо новий екземпляр форми редагування персонажа

CharacterEditForm editForm = new CharacterEditForm(selectedCharacter);

if (editForm.ShowDialog() == DialogResult.OK)

{

// Оновлюємо персонажа у списку

characters[index] = editForm.Character;

RefreshCharactersList();

lstCharacters.SelectedIndex = index;

}

}

// Обробник подiї натискання кнопки "Видалити персонажа"

private void btnDeleteCharacter\_Click(object sender, EventArgs e)

{

if (lstCharacters.SelectedItem == null)

{

MessageBox.Show("Будь ласка, виберiть персонажа для видалення!", "Попередження",

MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

// Запитуємо пiдтвердження видалення

Character selectedCharacter = lstCharacters.SelectedItem as Character;

DialogResult result = MessageBox.Show($"Ви дiйсно бажаєте видалити персонажа '{selectedCharacter.Name}'?",

"Пiдтвердження видалення", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (result == DialogResult.Yes)

{

// Видаляємо персонажа зi списку

characters.Remove(selectedCharacter);

RefreshCharactersList();

}

}

**Додаток Д.**

// Метод для збереження персонажiв у JSON форматi

private void SaveToJson()

{

try

{

// Налаштування серiалiзацiї JSON

JsonSerializerOptions options = new JsonSerializerOptions

{

WriteIndented = true,

PropertyNamingPolicy = JsonNamingPolicy.CamelCase

};

// Серiалiзуємо список персонажiв у JSON

string jsonString = JsonSerializer.Serialize(characters, options);

// Записуємо у файл

File.WriteAllText(JSON\_FILE\_PATH, jsonString);

MessageBox.Show("Персонажi успiшно збереженi у форматi JSON!", "Iнформацiя",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (Exception ex)

{

MessageBox.Show($"Помилка при збереженнi персонажiв у JSON: {ex.Message}", "Помилка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

// Метод для збереження персонажiв у XML форматi

private void SaveToXml()

{

try

{

// Створюємо XML серiалiзатор

XmlSerializer serializer = new XmlSerializer(typeof(List<Character>));

// Створюємо файловий потiк

using (FileStream fs = new FileStream(XML\_FILE\_PATH, FileMode.Create))

{

// Серiалiзуємо список персонажiв у XML

serializer.Serialize(fs, characters);

}

MessageBox.Show("Персонажi успiшно збереженi у форматi XML!", "Iнформацiя",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (Exception ex)

{

MessageBox.Show($"Помилка при збереженнi персонажiв у XML: {ex.Message}", "Помилка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

// Обробник подiї натискання кнопки "Завантажити персонажiв"

private void btnLoadCharacters\_Click(object sender, EventArgs e)

{

// Створюємо дiалог вибору формату завантаження

DialogResult result = MessageBox.Show("Завантажити з JSON файлу? (Якщо 'Нi', буде використано XML файл)",

"Вибiр формату", MessageBoxButtons.YesNoCancel, MessageBoxIcon.Question);

if (result == DialogResult.Cancel)

return;

try

{

if (result == DialogResult.Yes)

{

// Завантажуємо з JSON

if (!File.Exists(JSON\_FILE\_PATH))

{

MessageBox.Show($"Файл {JSON\_FILE\_PATH} не знайдено!", "Помилка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

string jsonString = File.ReadAllText(JSON\_FILE\_PATH);

// Важливо: використовуємо тi самi налаштування серiалiзацiї, що й при збереженнi

JsonSerializerOptions options = new JsonSerializerOptions

{

PropertyNamingPolicy = JsonNamingPolicy.CamelCase

};

characters = JsonSerializer.Deserialize<List<Character>>(jsonString, options);

MessageBox.Show("Персонажi успiшно завантаженi з JSON!", "Iнформацiя",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

// Завантажуємо з XML

if (!File.Exists(XML\_FILE\_PATH))

{

MessageBox.Show($"Файл {XML\_FILE\_PATH} не знайдено!", "Помилка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

XmlSerializer serializer = new XmlSerializer(typeof(List<Character>));

using (FileStream fs = new FileStream(XML\_FILE\_PATH, FileMode.Open))

{

characters = (List<Character>)serializer.Deserialize(fs);

}

MessageBox.Show("Персонажi успiшно завантаженi з XML!", "Iнформацiя",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

// Оновлюємо список персонажiв

RefreshCharactersList();

}

catch (Exception ex)

{

MessageBox.Show($"Помилка при завантаженнi персонажiв: {ex.Message}", "Помилка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}