МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ**

| | |
|---|---|
| ННІ/факультет | Інформаційних технологій |
| Кафедра | Інформатики і прикладного програмного забезпечення |
| Спеціальність | Інженерія програмного забезпечення |
| Форма навчання | Денна |

# КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА[1]

Носур Данило Михайлович
*(прізвище, ім'я, по батькові здобувача)*

на тему Розробка гри у жанрі RPG
*(повна назва теми)*

за матеріалами праць провідних спеціалістів з розробки ПЗ та проектування БД
*(повна назва бази дослідження)*

науковий керівник к.е.н., доцент _____ Баран С.В.
*(наук. ступінь, вчене звання)* *(підпис)* *(прізвище, ініціали)*

**Робота допущена до захисту в ЕК**
Протокол засідання кафедри
від 11.06.2025р. № 12
Завідувач кафедри _____
(підпис)
д.т.н., професор Зеленський О.С.
*Наук. ступень, вчене звання* *Ініціали, прізвище*

Кривий Ріг – 2025

[1] Розробка гри у жанрі RPG

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ**

| | |
|---|---|
| ННІ/факультет | Інформаційних технологій |
| Кафедра | Інформатики і прикладного програмного забезпечення |
| Спеціальність | Інженерія програмного забезпечення |
| Форма навчання | Денна |

«ЗАТВЕРДЖУЮ»
Завідувач кафедри _____ Зеленський О.С.
(підпис) (Прізвище, ініціали)
«11» червня 2025 року

# ЗАВДАННЯ
# НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ[2]

1. Тема роботи «Розробка гри у жанрі RPG»

Керівник роботи к.е.н., доцент Баран С. В.
затверджені наказом закладу вищої освіти від «04» квітня 2025 р. № 222-ст

2. Строк подання здобувачем роботи до «09» червня 2025 р.

3. Зміст кваліфікаційної роботи, об’єкт, предмет та мета дослідження:

**Розділ 1.** Постановка задачі

**Розділ 2.** Розробка алгоритму розв’язання задачі

**Розділ 3.** Розробка бази даних задачі

**Розділ 4.** Розробка програмного забезпечення

*Об’єкт дослідження: гра у жанрі RPG*

*Предмет дослідження: гра*

*Мета кваліфікаційної роботи: створенння гри у жанрі RPG*

5. Дата видачі завдання «04» квітня 2025 р.

[2] Розробка гри у жанрі RPG

**ЗГОДА здобувача вищої освіти**

Державного університету економіки і технологій про перевірку кваліфікаційної роботи на прояви академічного плагіату
та розміщення в Репозитарії Університету

Я, Носур Данило Михайлович (ПІП), підтримую політику Державного університету економіки і технологій з академічної доброчесності і відкритого доступу.

Засвідчую, що кваліфікаційна бакалаврська робота «Розробка гри у жанрі RPG» виконана самостійно та не містить академічного плагіату. Я не надавав і не одержував недозволену допомогу під час підготовки цієї роботи. Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Із чинним Положенням про запобігання та виявлення академічного плагіату в роботах здобувачів вищої освіти Державного університету економіки і технологій ознайомлений. Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення норм академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

Також я поінформований, що відповідно до «Положення про Репозитарій Державного університету економіки і технологій» зазначена робота буде розміщена в Електронному архіві Університету. З умовами такого розміщення ознайомлений(на).

Дата підпис ініціали, прізвище (власноруч)

# КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів МДР | Строк виконання етапів роботи | Відмітка керівника про виконання етапів (дата, підпис) |
|---|---|---|---|
| 1 | Підготовка розділу 1 | 04.04.2025-13.04.2025 | |
| 2 | Підготовка розділу 2 | 14.04.2025-26.04.2025 | |
| 3. | Підготовка розділу 3 | 27.04.2025-15.05.2025 | |
| 4 | Підготовка розділу 4 | 16.05.2025-08.06.2025 | |
| 5 | Реєстрація завершеної кваліфікаційної роботи | 09.06.2025 | Реєстраційний №___ «09»червня 2025 р. |
| 6 | Отримання відгуку від наукового керівника | 10.06.2025 | |
| 7 | Подання кваліфікаційної роботи на перегляд завідувачу кафедри | 11.06.2025 | |
| 8 | Отримання зовнішньої рецензії | 12.06.2025 | |
| 9 | Попередній захист кваліфікаційної роботи на кафедрі | 13.06.2025 | |
| 10 | Підготовка до захисту в ЕК | 16.06.2025-21.06.2025 | |

Завдання підготував науковий керівник _____ _____Баран С. В____
_
(підпис) (прізвище та ініціали)

Завдання одержав _____ _____Носур Д. М____
_
(підпис) (прізвище та ініціали)

# АНОТАЦІЯ

**на бакалаврську роботу**

**«Розробка гри у жанрі RPG»**

**Носур Д. М.**

Кваліфікаційна робота за спеціальністю 121 – Інженерія програмного забезпечення – Державний університет економіки і технологій. – Кривий Ріг, 2025.

У кваліфікаційній роботі розроблено гру у жанрі RPG на мові програмування C++ з використанням графічного API OpenGL для побудови візуального середовища гри та анімації персонажів.

Система побудови та збереження ігрового світу гри побудована за допомогою XML редактору карт TiledMapEditor, котра дозволяє побудовувавти та зберігати ігрову мапу та вміст ігрового світу.

За для вивантаження та відображення ігрової мапи було створено набір інструментів для зчитування та рендеру мапи.

У підсумку кваліфікаційної роботи розроблено функціональну комп'ютерну гру у жанрі RPG, що задовольняє основні вимоги до ігрового процесу, графічного оформлення, системи збереження та зручності використання.

Ключові слова: АНІМАЦІЯ, C++, OpenGL, OpenAL, JSON, C#, Windows Forms, RPG, ГРА.

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

| Позначення | Пояснення |
|---|---|
| **C++** | Мова програмування, яка поєднує процедурну та об'єктно-орієнтовану парадигми. Використовується для створення високопродуктивного програмного забезпечення, зокрема ігор. |
| **OpenGL (API)** | Графічний інтерфейс програмування, що забезпечує прямий доступ до графічного процесора для відображення 2D- та 3D-графіки. |
| **Windows Forms** | Платформа для створення графічних інтерфейсів у середовищі Windows на базі .NET. |
| **RPG (Role-Playing Game)** | Жанр комп’ютерних ігор, у яких гравець керує персонажем, виконує завдання, досліджує світ та впливає на розвиток сюжету. |
| **NPC (Non-Player Character)** | Персонаж, що не контролюється гравцем. Може бути ворогом, союзником або нейтральним об’єктом. |
| **A*** | Алгоритм пошуку найкоротшого шляху в графі. Використовується для навігації персонажів у грі. |
| **TileMap** | Система побудови карти з окремих плиток (тайлів). Використовується для створення рівнів у 2D-іграх. |
| **Shaders** | Невеликі програми, що виконуються на GPU для обробки графіки (освітлення, тіні, кольори тощо). |
| **RenderInstance** | Базовий клас для рендеру об'єктів у грі, що керує буферами та геометрією на рівні OpenGL. |
| **Content** | Каталог, у якому зберігаються всі ресурси гри: текстури, мапи, звуки, скрипти, UI, анімації тощо. |
| **XML** | Формат структурованих текстових даних, що використовується для збереження мап, властивостей, тайлів тощо. |
| **UI (User Interface)** | Користувацький інтерфейс — візуальні елементи, з якими взаємодіє гравець (меню, кнопки, панелі тощо). |

**ЗМІСТ**

# ВСТУП

Сучасна індустрія комп’ютерних ігор стрімко розвивається, охоплюючи дедалі ширше коло користувачів та інтегруючись у різні сфери життя – від розваг до освіти, науки й психології. Ігри перетворилися на повноцінний культурний і технологічний феномен, що поєднує програмування, графічний дизайн, сценарну майстерність і моделювання складних віртуальних світів. Особливе місце серед численних жанрів займають ігри жанру RPG (role-playing game) – рольові ігри, що дозволяють користувачу не просто брати участь у подіях, а й втілюватися у вигаданого персонажа, впливати на розвиток сюжету, розвивати навички героя та досліджувати ігровий світ.

Аналітики стверджують, що загальні прибутки кіноіндустрії та музичної індустрії вже перевищили прибутки відеоігор. Це викликає високий попит на фахівців, які можуть створювати високоякісний ігровий контент. Крім того, у технічних університетах дедалі частіше використовується розробка ігор як навчальний інструмент. У цих університетах курсові та дипломні проекти базуються на ігрових рушіях.

Створення ігор такого жанру потребує глибоких знань в об'єктно-орієнтованому програмуванні, розробці інтерфейсу користувача, побудові ігрової логіки, збереженні даних та відображенні графіки у реальному часі. Особливої популярності набувають RPG-ігри з відкритим світом, у яких гравець отримує можливість вільного пересування локаціями, взаємодії з елементами середовища, виконання завдань у довільному порядку та прийняття рішень, що безпосередньо впливають на розвиток подій у грі. Такий підхід суттєво збагачує геймплей, надаючи йому більшу гнучкість, різноманітність і занурення у віртуальний світ.

Для створення таких ігор необхідно володіти такими предметами, як математики (лінійна алгебра та геометрія), фізики (для реалізації взаємодії об’єктів) і основ дизайну користувацького досвіду. Особлива увага потрібна для створення внутрішньої ігрової системи, яка включає бойову механіку, інвентар, систему діалогів і квест.

У рамках даної кваліфікаційної роботи реалізовано розробку комп’ютерної гри у жанрі RPG з відкритим світом, яка створена мовою програмування C++ з використанням графічного API OpenGL. Використання технологій OpenGL забезпечує якісне відображення двувимірної та тривимірної графіки, включаючи реалізацію освітлення, тіней, анімації та інших візуальних ефектів, які є необхідними для формування правдоподібного ігрового простору. Також у проекті приділено увагу зручності користувацького інтерфейсу, можливості збереження ігрового прогресу, а також адаптації гри для встановлення на персональні комп’ютери.

Створення власної системи рендерингу та оптимізації графіки для стабільної роботи на звичайних ПК було однією з найцікавіших проблем у проєкті. Це дозволило глибше зануритися в структуру графічного пайплайну та зрозуміти, як досягається гармонії між продуктивністю та якістю візуалізації.

Проект було реалізовано за допомогою OpenGL, відкритого стандарту графіки, який дозволяє працювати з низькорівневими графічними можливостями апаратного забезпечення. Це покращило розуміння основ графічного рушія «з нуля» без використання готових ігрових платформ, таких як Unity або Unreal Engine.

Актуальність теми пояснюється не лише популярністю RPG-жанру серед гравців, а й тим, що розробка таких ігор дозволяє на практиці застосовувати широкий спектр знань у галузі програмної інженерії, комп’ютерної графіки.

Метою дипломної роботи є створення прототипу комп’ютерної гри у жанрі RPG, що забезпечує базові механіки гри у відкритому світі з використанням сучасних інструментів розробки графіки та звуку. У процесі роботи було проаналізовано особливості жанру, визначено ключові компоненти гри, реалізовано графічне середовище, механіку переміщення та взаємодії, а також забезпечено технічну можливість запуску проєкту на цільових пристроях.

У наші дні, коли розробка ігор стає легшою завдяки відкритим бібліотекам і фреймворкам, особливо важливо спробувати створити ігри з нуля. Це не лише

показує, наскільки добре ви володієте інструментами, але й допоможе вам краще зрозуміти, як будуються складні системи.

Таким чином, створення відкритого світу RPG — це не просто спосіб реалізації творчих ідей, а й повноцінне технічне завдання, яке охоплює одразу кілька напрямків прикладної інформатики. Програмування, архітектурне проектування, обробка графіки в реальному часі, робота з візуальними та звуковими ресурсами, оптимізація продуктивності та розробка інтерфейсів користувача – усе це складові цього проекту. Розробник повинен добре розуміти алгоритми, структури даних, принципи роботи з пам’яттю та графічний пайплайн під час процесу реалізації. Він також повинен вміти застосовувати це розуміння в реальному світі.

Крім того, виконання такого завдання значно покращує розуміння комп’ютерної графіки, особливо працюючи з OpenGL, однією з найвідоміших бібліотек для низькорівневого рендерингу. Підготовка спеціаліста, який планує працювати в сфері геймдеву або суміжних галузях, включає вивчення основ побудови сцени, шейдерів, обробки координат і матриць трансформацій.

Таким чином, створення відкритого світу RPG — це завдання, яке включає в себе інженерний підхід, творчість, системне мислення та інженерний підхід. Завдяки цьому можна розвивати як технічні навички, так і навички планування, командної роботи, самостійного прийняття рішень, проєктування та планування. Це не просто технічне досягнення; це також показує професійну зрілість і готовність до справжніх викликів у сфері сучасних IT.

# РОЗДІЛ 1
# ПОСТАНОВКА ЗАДАЧІ

## 1.1. Основні визначення та характеристика задачі

Розробка гри у жанрі RPG у відкритому всесвіті є завданням не з легких, та потребує від виконавця великої відповідальності та широко спектру знань як у сфері програмування із використанням графічного API OpenGL, впевнених знань мови програмування C++, так і орієнтуватися у ігрових технологіях, побудові високоякісного ігрового продукту, та розуміти базові ігрові механіки та типи ігрових жанрів.

В першу чергу при створенні гри такого жанру необхідно чітко вирішити основну тематику та ігровий сеттінг гри. Вже базуючись на цьому в подальшому необхідно із великою відповідальністю підійти до підбору ігрових асетів, на яких буде будуватись наша гра, та прописати базовий сценарій гри.

Після чіткої постановки задачі, наступним кроком, необхідно вирішити який стек технологій буде найбільш ефективним для вирішення поставленої задачі. Після чого, можна впевнено підходити до формування базової структури нашої гри.

В нашому випадку, як було сказано вище, гра буде реалізована із використанням графічного API OpenGL на мові C++ у жанрі RPG. Сеттінг гри я обрав середньовічний із вмістом магічних та міфічних створіннь, щось на кшталт сеттінгу таких ігор як “Відьмак”. Для ігор подібного жанру й сеттінгу у просторі інтернет є багато різноманітних асетів. Також, великим плюсом таких ігор є їх популярність, що надасть нам змогу заробити гарні гроші у випадку якісного створення гри та її монетизації у майбутньому.

Створення гри особливо такого жанру та масштабу - задача, котра потребує великого вмісту гнучкого інструментарію у арсеналі розробника. Тому, також, як однією з провідних задач при створенні цього дипломного проекту я вирішив створити набір інструментарію для створення двовимірних ігор, щось на кшталт ігрового двигуна, котрий у майбутньому спростить створення ігор, та надасть

змогу ефективно використовувати побудову та рендер графіки за допомогою OpenGL.

Таким чином, ми маємо перед собою наступні задачі:

1. В першу чергу, ми маємо створити якісний прототип гри у жанрі RPG, котрий міститиме ігрову мапу вільну для дослідження, та стабільну роботу гри з гарним оформленням та привабливою графікою та атмосферою. Гра має мати звукове супроводження. Сеттінг - середньовічно-міфічний, на кшталт серії ігор “Відьмак”.
2. Мапу гри та вміст контенту на ній ми маємо редагувати зручними сторонніми (або ж своїми) засобами, котрі потім ми матимемо змогу зчитувати та відрендерити і відобразити у грі.
3. За для зручного та ефективного створення гри необхідно створити зручний інструментарій, на кшталт ігрового двигуна, котрий буде спрощувати створення гри у довгостроковій перспективі.
4. Структура гри на інструментів для розробки має бути зручною та інтуїтивно зрозумілою для експлуатації іншими розробниками, що надасть змогу для створення та підтримки гри командою у майбутньому.
5. Гра має давати великий простір для розвитку та монетизації (наприклад простір для створення сиквелу, спін офу, фан-сервісу, онлайн режиму, та іншого виду розвитку у майбутньому).

## 1.2. Аналіз аналогів та конкуренції на ринку

Відверто кажучи, аналогів ігор подібного жанру і сетінгу на ринку є багато. Але, не варто втрачати ентузіазму, адже слідкуючи за тенденцією деградації сучасного ринку AAA ігор можна з повною упевненістю сказати, що інді ігри, особливо жанру RPG та такого сетінгу, є досить затребуваними серед геймерів усього світу.

Але, не варто недооцінювати конкуренцію на ринку, адже прихильників старої школи RPG у нас час також вистачає, й час від часу ринок дивує нас такими шедеврами ігрової індустрії, як, наприклад, нещодавно випущена Kingdome Come: Deliverance 2.

Kingdome Come: Deliverance 2 - це гра від чеських розробників Warhorse Studio, сюжет котрої розповідає нам про події, котрі відбулися у Богемії XV століття. Події гри мають історичний контекст із нотками домислів (як, наприклад, головний герой гри Індро).

Щойно гра з'явилася на світ, як в ту ж мить вона обзавелась величезною любов'ю геймерів з усього світу, як нових, так і старих ветеранів серії котрі чекали продовження цілих 7 років!

Рецепт успіху гри був досить простий - у керма досить молодої студії Warhorse був гарний сценарист Данієль Вавра, котрий раніше написав сюжети та створив такі світові хіти як “Мафія 1” та “Мафія 2”. Чоловіка в першу чергу цікавило не власне збагачення, а створення дійсно гарних ігор, котрі розповідають дійсно цікаві, зворушливі та інтригуючі історії! Саме такими були сюжети його ігор.

Також, велике значення для успіху гри відіграє її наративний дизайн, світ гри (наскільки він живий та наскільки він відображає те, що відбувається у грі), та як світ реагує на дії гравця. Підбиваючи підсумки сказаного вище, нам важливо створити світ, в якому справді гравцям хочеться жити!

Отже, таким чином, ми можемо зробити висновок, що не мало важливим аспектом у створенні гри є створення живого ігрового всесвіту, який реагує на дії користувача, та відображає основні мотиви та центральну історію, котру розповідає нам сюжет гри.

Також, задля більш комфортного та приємного ігрового процесу, необхідно зробити гарну оптимізацію для гри на різних пристроях, щоб навіть користувачі із не надто сильними персональними компьютерами могли грати у гру із комфортом! Досвід першої частини Kingdom Come: Deliverance показує як

ніяка інша гра, що користувачі дуже чутливі до оптимізації та працездатності ігрового продукту!

Не дивлячись на те, що Kingdom Come: Deliverance 2 - гра у трьохвимірному всесвіті, а моя гра націлена на двохвимірне, досвід створення користувацького інтерфейсу є єдиним, та не досить видозміненим серед ігор жанру RPG. Тож, розглянемо користувацький інтерфейс гри (рис. 1.1).

Рис. 1.1. Користувацький інтерфейс гри Kingdom Come: Deliverance 2

Як ми можемо спостерігати на рисунку 1.1, у верхній частині екрану ми можемо спостерігати радар, котрий вказує користувачеві на його поточне направлення, та направлення поточного квесту, котре виконує гравець.

Такий спосіб навігації є досить розповсюдженим серед ігор жанру RPG. Таким чином користувач буде більше заохочений до дослідження ігрового всесвіту, адже він не має конкретного маршруту , котрий веде його від точки А до точки Б коротшим шляхом, що спонукає гравця самому шукати шлях для досягнення точки.

Таким чином, прокладаючи шлях до поточного завдання користувач може наткнуться на багато інших цікавих місць на ігровій карті (наприклад другорядних квестів або просто цікавих місць для досліджень).

Також, на мою думку, задля більш безшовного дослідження ігрового всесвіту було б добре створити велику ігрову мапу котра буде підвантажуватись в залежності від поточного місцезнаходження користувача, не вдаючись до екранів завантаження між ігровими локаціями.

Не мало важливою частиною кожної RPG гри є також інвентар користувача. У Kingdom Come: Deliverance 2 розробники реалізували його наступним чином (рис. 1.2).


Рис. 1.2. Інвентар гравця у грі Kingdom Come: Deliverance 2

На мою суб'єктивну думку інвентар у грі реалізован не досить добре, адже він дуже перевантажений інформацією, та досить важко одразу зорієнтуватися що за що відповідає й куди звертатись для отримання потрібної тобі у моменті часу інформації.

Тут ми бачимо можливість переключитись серед пунктами ігрового меню (мапа, журнал подій, інвентар, тощо), кожен з яких призваний відображати свій особистий аспект інформації, будь то ігрова мапа або список взятих користувачем квестів.

У ігрового меню, котре ми зараз розглядаємо, є свої підпункти, котрі фільтрують вміст інвентаря користувача по типу вмісту (зброя, обладунки, їжа,

тощо), що э досить зручно якщо нам необхідно відфільтрувати вміст свого багажу, котрий у іграх подібного жанру загалом завжди досить переповненим вмістом.

Відфільтрований інвентар, в свою чергу, ми можемо відсортувати за назвою, ціною, якістю, тощо.

Згодом, обравши якийсь вміст інвентарю, ми можемо прочитати коротку інформацію щодо нього, переглянути його характеристики, та якщо це зброя або ж обладунки, то можемо переглянути чи є вони ліпшими або ж гіршими за наши поточні, котрі надів користувач на свого персонажа.

Врешті-решт, якщо зброя чи обладунки нам підходять, ми можемо застосувати їх до нашого персонажа (або ж виконати з предметом інвентаря якусь взаємодію, наприклад з`їсти, якщо це щось їстне).

Таким чином, досліджуючи інвентар гри Kingdom Come: Deliverance 2, ми можемо зробити висновок, що в першу чергу інвентар гравця має бути легким і інтуїтивно зрозумілим у експлуатації, надавати користувачеві коротку та вичерпну інформацію щодо елементу інвентаря, та надати користувачеві можливість сортувати його вміст за власними вимогами та потребами.

Також, інвентар не має бути переповнений великою кількістю інформації.

Оцінивши досвід великої ААА гри, котрою э Kingdom Come: Deliverance 2, я пропоную перейти до чогось більш приземленого та більш схожого по технічній частині до моєї гри.

Говорячи про двовимірні інді ігри у жанрі RPG неможливо не згадати Stardew Valley, котра не була зроблена великою компанію, а створена одним єдиним ентузіастом.

Дана гра виконана у досить привабливій та простій двовимірній піксельній стилістиці, та уособлює у собі двовимірну гру в жанрі RPG в котрій користувачеві необхідно доглядати за власною фермою!

Не дивлячись на свою простоту, гра заволоділа великою фан базою, та підтримується розробником по сьогодення!

Розглянемо основні аспекти завдяки яким гра стала такою популярною:

1. Простота та невибагливість до апаратних можливостей компьютера користувача. Так як гра є двовимірною, графіка якої будується завдяки спрайтам, гра є зовсім не вимогливою до потужностей. Вона запускається та комфортно працює на будь-якій системі! Таким чином навіть користувачі зі слабкими персональними компьютерами можуть насолодитися цією грою, що надає грі досить великий простір до здобуття фан бази.
2. Простота та неспішний ігровий процес. Якщо деякі ігри жанру RPG є швидкими, складними, та можуть вивести деяких користувачів із рівноваги, то Stardew Valley навпаки зачаровує користувача своїм примиренням та спокійною атмосферою. Гра створена саме для того, щоб користувач отримував насолоду, задоволення, розслаблення та відпочинок від буденності під час ігрового процесу. У грі ти просто доглядаєш за власним ранчо! Ідеальний медитативний ігровий процес котрий призваний заспокоїти та відволікти користувачів від буденності!
3. Приємне візуальне оформлення. Не дивлячись на те, що гра має піксельну двовимірну графіку, візуал гри є досить приємним та заспокійливим.

Розглянувши основні моменти котрі принесли грі популярність, ми можемо зробити наступні висновки:

1. Грі не обов’язково мати складку графіку котра включатиме в себе провідні технології, грі необхідно мати просто привабливий візуальний вигляд.
2. За для більшого охопу аудиторії, важливо щоб гра добре працювала на більшості системах та комплектуючих.
3. Важливо щоб гра була не досить складною, щоб користувач почував себе в ній комфортно та розслаблено.

Розглянемо візуальне оформлення гри (рис. 1.3).


Рис. 1.3. Візуальне оформлення гри Stardew Valley

Як ми можемо спостерігати, графіка гри є не складною, але в ту ж саму мить досить приємною.

Користувацький інтерфейс складається з панелі швидкого доступу користувача до предметів інвентаря (нижня панель), що надаватиме користувачеві змогу швидкого доступу до будь яких з обраних раніше елементів.

Такий підхід є також досить популярним серед ігор подібного жанру, але в нашому випадку доречніше буде обрати підхід котрий був у Kigdom Come: Deliverance 2. Там також було таке меню швидкого доступу, але до нього можна було додати невелику кількість елементів, кількість яких також базувалась в залежності від того який пояс надітий на користувача.

Обираючи такий підхід до реалізації меню швидкого доступу ми отримаємо можливість надати користувачеві більше простору для побудови власного типу персонажа.

Розглядаючи зображення 1.3, ми також можемо помітити відсутність будь-якої навігації, але ми можемо помітити датчик котрий демонструє поточний час доби у грі та кількість грошей, котрі наявні на рахунку гравця.

Такий підхід зумовлений тим, що гра більш акцентує увагу на процесі догляду користувача за власним господарством, дорогу до якого користувач завжди пам'ятає. Гра не має як такої системи квестів, як інші ігри жанру RPG, та великого ігрового всесвіту, тому і великої потреби у системі навігації у гри не має.

Не менш важливим аспектом ігор подібного типу є діалоги із персонажами та подача інформації.

Діалогова система гри Stardew Valley реалізована наступним чином (рис. 1.4).


Рис. 1.4. Візуальне оформлення гри Stardew Valley

Така система є досить розповсюдженою серед ігор подібного жанру та графічної реалізації, адже сворючи гру у піксельному форматі досить дивно було б чути справжній дубляж з вуст персонажів гри! Тому, такий підхід є досить оптимальним, та не викликає питанн та обурення серед гравців.

Вимогою до моєї гри у майбутньому буде створення системи діалогів та розгалуження ігрового сюжету та подій в залежності від вибору користувача в процесі гри.

Така система надасть нам не лише ефект занурення користувача у гру та її світ, але й відчуття впливу на неї.

Також, безспірним плюсом є можливість перепроходження гри у майбутньому задля отримання інших (можливо навіть унікальних) варіантів розвитку тих чи інших подій.

## 1.3. Формування вимог до продукту

Отже, розглянувши наявні на сучасному ринку ігор аналогів, можна зробити наступні висновки та вимоги до мого власного ігрового продукту:

1. В першу чергу необхідно створити аналог власного ігрового двигуна, котрий буде спрощувати створення та підртимку гри у довгостроковій перспективі.
2. Структура проекту та його код має бути зрозумілим, щоб інші розробники також могли зрозуміти код та зорієнтуватись у структурі проекту.
3. Проект має бути гарно оптимізований під нові та не дуже нові системи користувачів. Адже як ми зробили висновок на прикладі двох розглянутих мною ігор, працездатність та плавність роботи гри досить позитивно відгукується у користувачів. Також, досить не захмарні системні вимоги позитивно впливають на поширення гри серед геймерів.
4. Графіка гри та стилістичне оформлення має бути не складним, але досить приємним. Нині на ринку комп'ютерних ігор достатньо проектів із складними та прогресивними графічними технологіями. Мета цього проекту - здобути навички та покласти початок моєму шляху у сфері геймдеву, а аж ніяк не створення проривної та дорогої комп'ютерної графіки, котру і так нині можуть продемонструвати такі передові ігрові двигуни як Unity або ж Unreal Engine 5. Сучасний ринок комп'ютерних ігор наситився вдосталь гарними

графічними технологіями. Нині серед геймерів з усього світу у пріоритеті цікавий геймплей, цікава задумка та сюжет!

## ВИСНОВКИ ДО РОЗДІЛУ 1

Основні вимоги до розроблюваного ігрового продукту були визначені шляхом аналізу поточного ринку RPG-ігор і порівняння з існуючими аналогами. Основний акцент лежить на створенні власного ігрового рушія, який буде простим, але функціональним, щоб забезпечити гнучкість і легкість підтримки проєкту в майбутньому. Вкрай важливо дотримуватися стандартів чистої архітектури та зрозумілої структури коду, щоб інші розробники могли легко орієнтуватися в проекті.

Особлива увага приділяється оптимізації, оскільки гра повинна добре працювати на широкому спектрі апаратного забезпечення. Це сприятиме популяризації гри та підвищить її доступність. Графічна складова не повинна бути надмірно перевантаженою, оскільки її функцією є створення приємного візуального враження, а не демонстрація технологічних можливостей.

# РОЗДІЛ 2
# АЛГОРИТМИ РОЗВ'ЯЗУВАННЯ ЗАДАЧІ

## 2.1. Алгоритми створення та завантаження ігрової мапи

Задля створення повноцінного ігрового світу мені необхідно перевірене часом рішення для формування двовимірної мапи для створення зі спрайтів графіки моєї гри.

Моє рішення пало на використання TiledMapEditor, повноцінного спрайтового редактора ігрових мап, котрий дозволяє створювати мап різних позиціонування. В моєму випадку - вид на мапу в мене буде зверху до низу.

TiledMapEditor формує ігрову мапу у виді xml коду, котрий представлений у виді .tmx та .tsx файлів, де .tmx - це сама ігрова мапа, а .tsx - файл із данними для зчитування текстури із файлу.

За для зчитування мапи із файлів необхідно створити належний інструментарій для мого рушія. Тому, створивши інструмент рушія, повна структура якого має наступний вигляд (рис. 2.1).

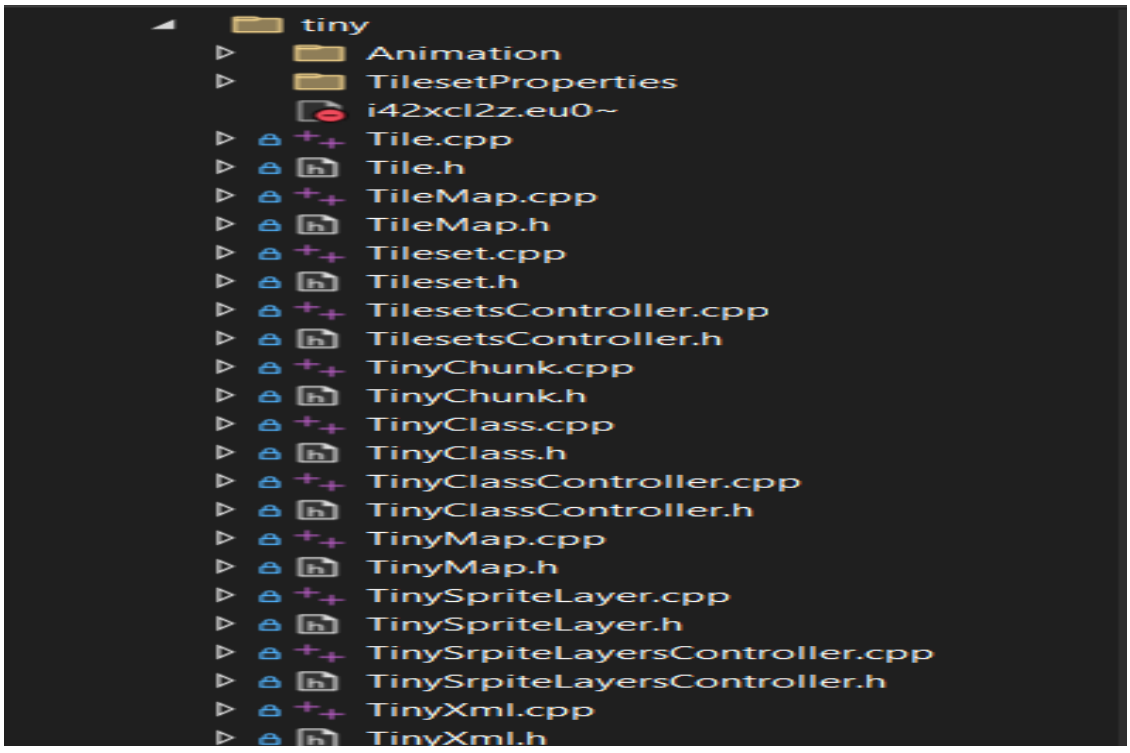


Рис. 2.1. Структура інструменту для зчитування TileMap мап

Для того, щоб картки, створені в редакторі Tiled, працювали, необхідно встановити систему обробки та зчитування файлів у форматі.tmx. Планується, що клас TinyXml, який запускає процес завантаження карти, стане основним компонентом цієї системи. Зокрема, запуск відбуватиметься за допомогою методу LoadMap, який виконуватиме початкову обробку вхідного файлу мапи.

На першому етапі передбачається провести перевірку наявності файлу. tmx і переглянути його структуру. Наступним кроком має бути зчитування кожного тайлсета, який є підключеним. Для цього потрібно використовувати клас TilesetsController, який оброблятиме структуру тайлів, включаючи шлях до зображення, розміри, типи та інші властивості. Контролер повинен зберігати зчитані тайлсети у зручному для використання форматі.

Далі планує створити систему для зчитування класів мапи, тобто глобальних шарів, які логічно групують різні об’єкти, такі як будинки, природні компоненти або функціональні зони. Це можна зробити за допомогою класу TinyClassController, який відповідатиме за створення структури шарів і визначення порядку їх відображення.

Наступним кроком має бути обробка спрайтових шарів, які є безпосередніми місцями розташування графічних елементів. На цьому етапі планується зчитувати такі елементи, як координати тайлів, анімації, властивості колізій, точки спавну, зони взаємодії тощо. Спрайтові шари є важливим компонентом візуального представлення карти, тому передбачити гнучку структуру обробки для їх зчитування.

Після обробки всіх даних карта повинна бути передана в TinyMap, об’єкт-контейнер. Цей клас матиме здатність виконувати функції контролера мапи, зберігаючи стан кожного елемента, координати об’єктів, інформацію про взаємодію, логіку оновлення та виведення на екран.

Створення похідного класу на основі TinyMap передбачається з метою впровадження спеціальної логіки для кожної ігрової мапи. Таким класом у цьому проекті має стати WonderWorld, у якому методи Update() і OnTriggerEnter() будуть змінені відповідно до сценаріїв гри.

Поліморфний і розширюваний підхід до побудови ігрового світу забезпечується за допомогою подібної структури побудови мапи. Такі архітектури дозволяють розробникам модифікувати логіку поведінки об’єктів, створювати нові типи локацій і реалізовувати взаємодії, не змінюючи ядро рушія. У майбутньому це дозволяє розширювати проєкт, додаючи нові мапи або сюжетні розгалуження, зберігаючи загальну стабільність системи.

## 2.2. Створення алгоритмів рендеру

Після завантаження ігрової мапи необхідно створити систему рендерингу, яка дозволить відображати ігрові об’єкти на екрані в режимі реального часу. Щоб забезпечити простий рендеринг статичних елементів і вивід складних анімованих об’єктів, така система повинна бути масштабованою, гнучкою та придатною до розширення.

Очевидно, що Rect стане основним класом для графічного представлення ігрових об’єктів. Ця група повинна виконувати функцію геометричної основи, тобто прямокутного об’єкта, на який можна накладати текстуру. Таким чином, кожен Rect має перетворюватися з простої фігури на повноцінний візуальний елемент, який може бути персонажем, декором, будівництвом або інтерфейсним компонентом.

У класі Rect слід передбачити методи для:

1. Встановлення розміру, місця та кольору.
2. Прив’язка матеріалу або текстури.
3. Використання миші (наведення, натискання).
4. Оновлення візуального стану та координат.

Це дозволяє Rect використовувати як універсальний будівельний блок, щоб вивести всі видимі елементи гри. Гнучкість цього класу повинна дати можливість легко розширити та адаптувати свою поведінку до конкретних завдань.

Паралельно з цим необхідно впровадити систему керування шейдерами, яка буде централізованою. Це вимагає створення класу ShadersController, який відповідатиме за:

1. Створювати та компілювати шейдери з файлів. верт і.frag.
2. Зберігання їх у контейнері unordered_map, який має унікальні ідентифікатори.
3. Управління шейдером під час рендерингу.
4. Передача уніформ-параметрів, таких як int, float, bool, vec2, vec3 та mat4, серед інших.

Завданням ShaderController є надання простого API для роботи з графічними ефектами, який дозволить змінювати візуальне оформлення об’єктів без втручання в внутрішню логіку рендерингу. Це дуже важливо для створення адаптивного освітлення, накладення тіней і візуальних змін на об’єктах, які змінюються залежно від стану гри чи гравця.

Розробка структури для підготовки геометричних та буферних даних є необхідною крім шейдерів. Для цього необхідно виконати клас RenderInstance. Його обов’язками будуть:

1. Створювати та зберігати буфери VAO, VBO та EBO.
2. Перегляду геометричних даних.
3. Передача їх для рендерингу на GPU.
4. Гарантувати, що логіка рендеру залишається окремою від логіки решти гри.

RenderInstance має бути реалізований як базовий клас, який легко адаптується до різних типів об’єктів. Залежно від ситуації він дозволить:

1. Швидко переходити між різними методами рендерингу.
2. Використовувати буфери знову і знову.
3. Підвищити продуктивність за допомогою оптимізації завантаження графічних даних.

Завданням взаємодії Rect, ShadersController і RenderInstance є створення єдиної модульної системи рендерингу, де кожен елемент має відповідати за свою

зону відповідальності, але все одно легко вписується в загальну архітектуру рушія. Таким чином можна реалізувати як прості сцени (користувачеві інтерфейси, меню), так і складні динамічні карти з великою кількістю об’єктів без втрати продуктивності.

Підводячи підсумок, рендеринг має бути адаптивним до вимог конкретної гри, щоб забезпечити стабільну візуалізацію незалежно від складності сцени.

## 2.3. Реалізація класу персонажів

Запропонований рушій має містити універсальний клас, який відповідатиме за керування та представлення ігрових персонажів, як і в будь-якій грі жанру RPG. Предбачається, що таким класом буде Pawn, який повинен бути основним для всіх живих істот у грі, незалежно від того, чи це гравець, ворог чи NPC.

Основні функції, необхідні для існування об’єкта в світі гри, включають логіку руху, обробку колізій, взаємодію з іншими об’єктами, відображення на екрані та реакцію на дії користувача, включаючи наведення курсора, кліки та події зіткнень. Цей клас повинен бути створений як основа для наслідування, щоб розробники могли створювати нові персонажі з унікальною поведінкою, не змінюючи основних функцій.

Згідно з цим методом, Pawn має мати такі основні характеристики:

1. Місцезнаходження та розмір об’єкта на мапі.
2. Швидкість руху, вагу, орієнтація.
3. Рівень здоров’я людини та ризики.
4. Бойові властивості, особливо радіус атаки.
5. Можливість взаємодіяти з іншими об’єктами та оточенням мапи.

Крім того, необхідно впровадити систему станів, яка визначає, як поводиться персонаж, будь то живий, керований гравцем, прихований або в кінематичному режимі.

Анімація, аудіоефекти, системи штучного інтелекту та обробники подій повинні бути підключені до класу. Реалізація абстрактних методів, таких як AIMovement(), Update() чи Draw(), які обов'язково перевизначають наслідувані класи, має бути особливістю архітектури Pawn. Це дозволяє розробити шаблон поведінки для кожного типу персонажа.

У майбутньому має бути реалізована ієрархія похідних класів, як-от:

1. Для ворожих персонажів, які рухаються по карті, відповідають гравцю та завдають шкоди;
2. Для користувача, який керує ігровим персонажем.

Крім того, необхідно створити інтерфейси для взаємодії з користувачем, такі як натискання, наведення миші та перевірка на перетин з різними об'єктами. Функції-хуки мають на меті надати можливість динамічно підключати власні обробники, які можуть використовувати кастомні відповіді без зміни ядра Pawn.

Важливо створити модульну структуру, яка розділяє всі важливі елементи (анімація, рендер, колізії, звук і матеріали) окремо. Це дозволить не лише спростити розширення функціональності, але й зробити повторне використання елементів у різних аспектах гри більш можливим.

Таким чином, клас Pawn має стати основним компонентом системи персонажів ігрового рушія. Його виконання повинно забезпечити гармонію між:

1. Простим процесом створення нових істот.
2. Підтримувати складну поведінку.
3. Стабільний рівень продуктивності рушія.
4. Потенціал для розширення проєкту в майбутньому.

## ВИСНОВКИ ДО РОЗДІЛУ 2

У цьому розділі описано стратегію реалізації основних алгоритмів і архітектурних компонентів, необхідних для створення RPG-гри, яка використовує OpenGL. Особлива увага приділяється розробці конструкції рушія, яка має бути побудована за принципом об'єктно-орієнтованого модульного дизайну, щоб дозволити ефективно керувати ігровими об'єктами та ресурсами.

Клас Pawn має стати основним компонентом системи персонажів, оскільки він відповідає за логіку руху, анімацію, взаємодію з середовищем, рендеринг і обробку подій. Його дизайн має підтримувати методи наслідування та перевизначення, щоб дозволити створювати дочірні класи з різними поведінками, такими як гравці, NPC та вороги. Гнучкість є життєво важливою для використання повноцінної RPG-логіки.

Це також передбачає впровадження класів Rect, TinyMap і WonderWorld. Для візуального представлення об’єктів базова геометрична одиниця має бути клас Rect. WonderWorld є конкретною реалізацією ігрового світу з відповідними сценаріями та елементами взаємодії, а TinyMap служить універсальним контейнером для всієї логіки мапи.

Контролери ресурсів повинні відігравати окрему роль у структурі рушія, зокрема:

1. TilesetsController — інструмент для керування тайлсетами.
2. TinyClassController — це пристрій, призначений для зчитування логічних шарів мапи.
3. TinySpriteLayersController — це програмний продукт, призначений для керування спрайтовими елементами.

Ці компоненти повинні працювати з даними у файлах.tmx і.tsx, щоб редактор Tiled міг обробляти всі вхідні структури.

Клас шейдерних контролерів є ще одним важливим компонентом системи. Він призначений для зміни алгоритму роботи з OpenGL-шейдерами. Він відповідає за зручне керування шейдерами, активацію, передачу параметрів і повторне використання. У поєднанні з ним клас RenderInstance відповідає за підготовку буферів (VAO, VBO, EBO) і рендеринг графічних об’єктів, що робить його універсальним шаблоном для будь-якого об’єкта сцени.

Таким чином, передбачувана система класифікації створює міцну основу для реалізації повноцінної RPG-гри, демонструючи ефективне використання технологій OpenGL і слугуючи початковим шаблоном для розробки інших 2D-ігор.

# РОЗДІЛ 3
# ОРГАНІЗАЦІЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1. Загальна характеристика інформаційного забезпечення

Інформаційне забезпечення ігрового рушія формує зміст гри. У ігровому проєкті вхідна інформація зберігається за допомогою файлової структури, об’єднаної у папці Content. Це відрізняє його від традиційних інформаційних систем, де дані зберігаються в базах даних, таких як Microsoft Access або MySQL.

Папка даних є основним сховищем даних про гру, які належать до логіки конкретної гри та не пов’язані з рушієм. У ньому є всі необхідні компоненти, включаючи мапи, зображення, анімації, шрифти, звукові ефекти, елементи інтерфейсу та багато іншого. Контролери рушія зчитують ці дані, які завантажуються під час запуску гри та використовується для створення інтерактивного ігрового світу.

Файлова система має наступну структуру (таблиця 3.1).

Таблиця 3.1

**Структура папки “Content”**

| № | Назва каталогу | Призначення |
|---|---|---|
| 1 | Animations | Містить файли анімацій персонажів |
| 2 | Assets | Загальні ресурси, зокрема шаблони, базові матеріали |
| 3 | Fonts | Шрифти, що використовуються в UI |
| 4 | Images | Зображення, спрайти, текстури |
| 5 | Maps | Ігрові карти у форматах .tmx, .tsx |

Продовження таблиці 3.1

| № | Назва каталогу | Призначення |
|---|---|---|
| 6 | Saves | Збереження ігрового процесу (власний формат, обробляється класом Saves) |
| 7 | Scripts | Ігрові сценарії та логіка (у форматі .txt, .lua, .cpp) |
| 8 | Sounds | Аудіофайли: музика, ефекти |
| 9 | UI | Елементи користувацького інтерфейсу |
| 10 | Winods | Кастомні вікна та модулі інтерфейсу (окремі класи) |

Як ми можемо спостерігати із представленої вище таблиці 3.1, структура папки Content має чіткий розподіл серед ігрових компонентів.

Такий підхід відокремлює ігровий вміст від логіки рушія, що дозволяє керувати його різноманітно. Кожен підкаталог містить певний тип ресурсів, що полегшує обробку за допомогою відповідних класів-контролерів, таких як TinyXml, TilesetsController, TinyMap, AudioController і ShaderController, серед інших.

Інформація з контенту використовується під час початку гри та її динамічного виконання. Таким чином, ця папка виконує функцію інформаційної бази даних проєкту, хоча вона зберігається у файловому форматі, який нестандартний.

## 3.2. Структура інформаційних масивів гри

Структура, яка забезпечує збереження ігрового середовища, об’єктів, колізій, точок спавну та інших елементів, є важливою частиною процесу

розробки гри RPG. Для цього проект використовує формат tmx — це XML-файл, створений Tiled Map Editor. Він містить повну інформацію про ігрову карту, включаючи шари, об’єкти, зв’язки з тайлсетами та типи колізій.

Файл карти world.tmx містить структуровану інформацію про всі компоненти карти, що дозволяє йому вважатися своєрідною базою даних ігрового середовища. користь від використання формату Його відкритість, розширюваність і простота інтеграції з XML-парсерами є перевагами tmx. У проєкті використовується ряд класів, таких як TilesetsController, TinyMap, TinyClassController і TinyXml, для зчитування та обробки цього файлу.

Нижче представлено короткий відрізок файлу ігрової мапи world.tsx (рис. 3.1).

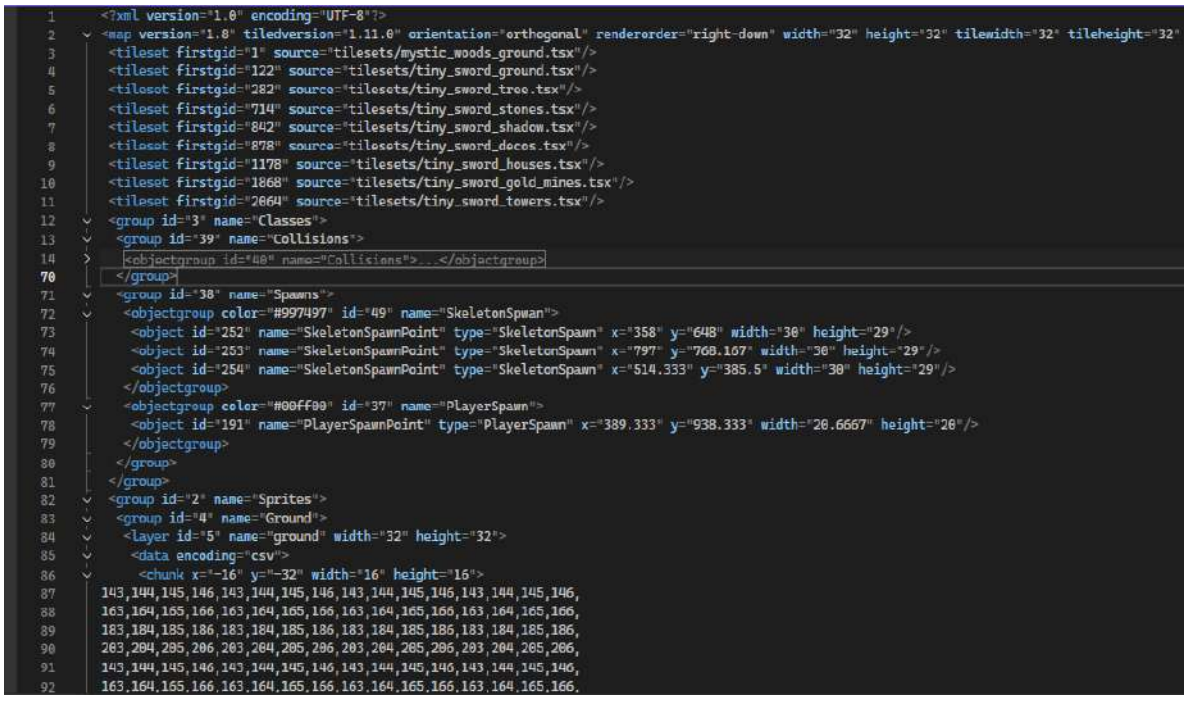


Рис. 3.1. Структура файлу world.tmx

Як можна спостерігати, у верхній частині файлу спочатку визначаються тайлсети, котрі надалі ми будемо використовувати для вказівок на ігровій мапі.

Тайлсети мають параметр firstgid, котрий вказує нам початковий індекс, з котрого починаються усі тайли, котрі використовуються для вказівки на мапі.

Наступна таблиця 3.2 продемонструє та детально опише теги, котрі використовуються у .tmx файлі.

Таблиця 3.2

**Теги .tmx файлу**

| XML-тег | Призначення |
|---|---|
| <map> | Основна обгортка карти: задає розміри, тип тайлів, режим рендеру |
| <tileset> | Підключення зовнішніх тайлсетів (наприклад, дерева, будівлі, тінь) через .tsx |
| <group> | Групування шарів за функціональністю (наприклад, Collisions, Spawns) |
| <objectgroup> | Список об’єктів: колізії, точки спавну тощо |
| <object> | Конкретний об'єкт на карті з типом, координатами, розміром |

Отже, детально оглянувши Таблицю 3.2, ми можемо перейти до огляду тайлсетів, котрі мають наступну структуру (рис. 3.2).


Рис. 3.2. Структура файлу тайлсету tiny_sword_tree.tsx

Із файлу тайлсету на рисунку 3.2, ми можемо спостерігати, що представлений на даному зображенні тайлсет має також параметри анімації, котрі ми також можемо створювати та налаштовувати у TiledMapEditor-i.

Також, поміж іншого, тайли даного тайлсету використовують налаштування (property тег) zIndex, котрі у подальшому будуть використовуватись для реалізації слоїв.

Розглянемо структуру тайлсету у наступній таблиці (таблиця 3.3).

Таблиця 3.3

**Структура атрибутів тайлсетів**

| **Атрибут** | **Значення** | **Опис** |
|---|---|---|
| tile id="2" | Цифра ID тайла | Унікальний ідентифікатор у тайлсеті |
| <properties> | zIndex = 1 | Вказує на порядок накладання |
| <animation> | набір <frame> | Анімаційна послідовність тайла |
| <frame tileid="2" duration="300"/> | кадр 1 | Відображає тайл з ID=2 протягом 300 мс |

Оглянувши таблицю 3.3, ми можемо зрозуміти, що даний підхід реалізації із використання тайлестів дозволяє реалізовувати анімаційні дерева, воду, світло тощо — не через окремий код, а через дані, що легко змінюються в редакторі.

Отже, для себе ми можемо відокремити наступні переваги використання .tmx/.tsx файлів:

1. Зручне редагування в Tiled без потреби перекомпіляції рушія.
2. Гнучка структура: можна додавати свої властивості (type, collision, zIndex) без зміни рушія.
3. Інтеграція з XML-контролерами: легко парсити через TinyXml, TilesetsController.
4. Підтримка груп, шарів і об’єктів — дозволяє розділяти логіку карти.
5. Масштабованість: підтримка нескінченних карт через infinite="1".
6. Підтримка анімації та поведінки напряму в тайлах.

Таким чином, файли.tmx і.tsx є повноцінними інформаційними масивами, які складають ігрове середовище, та які можна зручно маштабувати та видозмінювати у будь який момент, використовуючи TiledMapEditor. Уся ця структура інтегрується безпосередньо в рушій, що дозволяє легко змінювати рівні, додавати об’єкти та змінювати логіку без зміни коду програми.

## ВИСНОВКИ ДО РОЗДІЛУ 3

У цьому розділі розглядається організація інформаційного забезпечення програмного продукту. Зокрема розглядаються аспекти структурування ігрових ресурсів і зберігання даних у форматах, які зручні для геймдев-розробки.

Проаналізовано структуру папки контенту, яка є основним сховищем усіх ресурсів, необхідних для роботи гри. Такий метод дозволяє чітко відокремити рушійну логіку від контенту конкретної гри. Це дозволяє підтримувати, масштабувати та повторно використовувати рушійну логіку на інших проектах.

Увага приділяється структурі ігрових карт у форматах.tmx і.tsx. Інформаційні масії можуть функціонувати в цих XML-файлах, які містять інформацію про карту, об’єкти, шари, колізії, анімацію та властивості кожного елемента. З їх допомогою можна легко керувати ігровим середовищем, не переписуючи код; зміни можна вносити безпосередньо через редактор Tiled.

Використання такої моделі інформаційного забезпечення збільшує продуктивність розробки, покращує керованість проєктом і забезпечує масштабованість, оскільки зміни в ігрових об’єктах або мапах можуть бути впроваджені швидко, не порушуючи архітектуру рушія.

Таким чином, система інформаційного забезпечення ефективна, логічно структурована та повністю адаптована до вимог сучасного ігрового проекту.

# РОЗДІЛ 4
# РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗАДАЧІ

## 4.1. Загальна характеристика інформаційного забезпечення

Отже, врахувавши всі необхідні аспекти, необхідні нам для розробки вище поставленої задачі, можна впевнено підходити до створення власного ігрового продукту!

В першу чергу при запуску гри у нас відбувається зчитування даних щодо збереження гри, ігрової мапи, завантаження та компіляція шейдерів, генерація першочергових даних для ігрових об’єктів нашої мапи.

Таким чином, за для того, щоб гравець не очікував завантаження дивлячись на чорний екран, було створено екран завантаження гри (рис. 4.1).


Рис. 4.1. Екран завантаження гри

Такий підхід до екрану завантаження гри є не унікальним, а досить розповсюдженим серед ігрових розробників з усього світу!

Як ми спостерігаємо із рисунку 4.1, екран містить центральне зображення (котре, до речі, не є статичним, а змінюється в процесі завантаження гри), на фоні

якого ми маємо progress bar, котрий демонструє процент завантаження гри, під яким, у свою чергу, будуть розміщені підказки та поради стосовно гри (нині там розміщена фраза-заглушка).

Також, тут ми можемо спостерігати кастомні шрифти, котрій мій рушій також може завантажувати та обробляти.

Саме завдяки злагодженій роботі інструментів завантаження шрифтів, класу контролеру шейдерів, класів рендеру та модулю із кастомними ігровими ассетами (папка Content, у котрій розміщені скрипти вікна із завантаденням), все це в купі дозволяє створити цей екран завантаження гри!

Таким чином, використовуючи даний екран завантаження гри, в процесі підготовки всіх ігрових елементів, гравець може насолодитися гарними зображеннями та прочитати поради, котрі йому можуть бути корисними під час ігрового процесу!

Після успішного завантаження гри, гравець сможе спостерігати дивовижні краєвиди ігрового всесвіту, котрі може нам продемонструвати мапа WonderWorld (рис. 4.2).

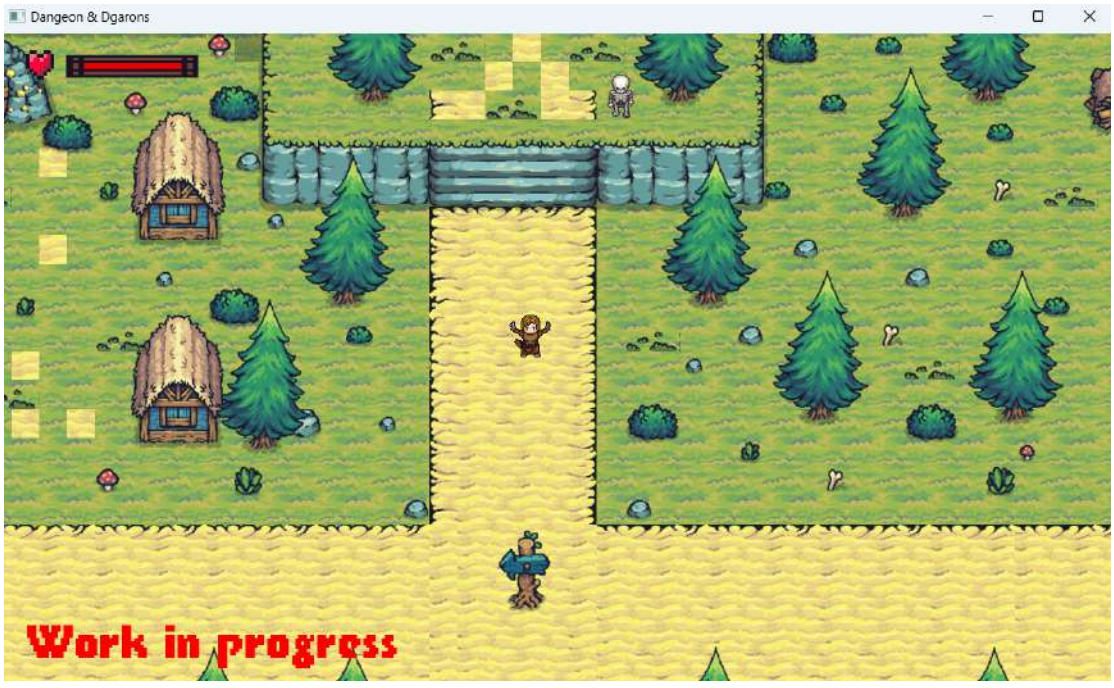


Рис. 4.2. Локація WonderWorld

Як можна бачити на рисунку 4.2, ігрова локація, що нам продемонстрована, має ліс, дерева якого програють анімацію (котра була зчитана

із .tsx файлу, продемонстровано вище), має дорогу, хати, 1-го вороже налаштованого NPC (скелет), котрий патрулює територію.

Як можна спостерігати, гравець та скелет також мають ігрові анімації.

За зчитування ігрової мапи відповідає класс TinyXml. Далі наведено таблицю з коротким описом змінних та методів класу TinyXml (таблиця 4.1).

Таблиця 4.1

**Змінні та методи класу TinyXml**

| **Назва** | **Тип** | **Розмір (байт)** | **Призначення** |
|---|---|---|---|
| root_dir | static std::string | ~32 | Статична змінна для зберігання кореневого шляху до XML-файлів |
| ReadDoc | static tinyxml2::XMLError | 4 | Зчитує XML-документ з файлу |
| LoadMap | static std::unique_ptr<TileMap> | ~8 | Завантажує карту з XML, створює об'єкт TileMap |
| ParsePolygon | static std::vector<Coord> | ~24 | Перетворює текстове представлення полігону у вектор координат |
| GetPathToTileSource | static std::string | ~32 | Формує шлях до ресурсу з тайлами |

Як було сказано вище (у пункті 2.1), наступним кроком після читання ігрової мапи є зчитування її тайлсетів, за які відповідає класс TilesetsController.

Тайлсети є незамінною та однією з найважливіших речей котрі необхідні за для зчитування ігрової мапи.

Отримати більш детальне представлення про клас TilesetsController нам надасть опис змінних та методів класу, представлених у таблиці нижче (Таблиця 4.2).

Таблиця 4.2

**Змінні та методи класу TilesetsController**

| Назва | Тип | Розмір (байт) | Призначення |
|---|---|---|---|
| tilesets | std::vector<std::shared_ptr<Tileset>> | ~24 | Вектор вказівників на тайлсети |
| TilesetsController() | Конструктор | - | Створює порожній контролер |
| TilesetsController(...) | Конструктор | - | Ініціалізують контролер тайлсетів з різних джерел |
| ~TilesetsController() | Деструктор | - | Звільняє ресурси |
| LoadTilesets | static void | 0 | Статичний метод зчитування тайлсетів з XML |

За зчитування та зберігання наших класів відповідає класс TinyClassController.

За для більшого розуміння структури класу TinyClassController, необхідно розглянути наступну таблицю 4.3.

Таблиця 4.3

**Змінні та методи класу TinyClassController**

| Назва | Тип | Розмір (байт) | Призначення |
|---|---|---|---|
| classes | std::unordered_map<int, std::shared_ptr<TinyClass>> | ~56 | Мапа об'єктів класів за ID |
| TinyClassController() | Конструктор | - | Створює порожній контролер |

Продовження таблиці 4.3

| Назва | Тип | Розмір (байт) | Призначення |
|---|---|---|---|
| TinyClassController(...) | Конструктор | - | Ініціалізує контролер даними або XML |
| ~TinyClassController() | Деструктор | - | Звільняє ресурси |
| LoadClasses | static void | 0 | Статичне зчитування класів з XML |

Вирішальним етапом у формуванні нашої ігрової мапи є зчитування слоїв тайлів.

Якщо класи - це більш глобальні слої, то спрайтові слої, більш локальні.

Класи можуть зберігати та відображати контекст того, що вони в себе вміщують (наприклад, дерева, будинки, дорогу, тощо), то спрайтові слої зберігають саме розміщення тих чи інших спрайтів на ігровій мапі.

Більш детально оглянути клас нам допоможе таблиця 4.4.

Таблиця 4.4

**Змінні та методи класу TinySpriteLayersController**

| Назва | Тип | Розмір (байт) | Призначення |
|---|---|---|---|
| spriteLayers | std::unordered_map<int, std::shared_ptr<TinySpriteLayer>> | ~56 | Мапа спрайтових шарів за ID |
| spriteLayerOrders | std::vector<int> | ~24 | Послідовність відображення шарів |
| TinySrpiteLayersController() | Конструктор | - | Створює порожній контролер |
| TinySrpiteLayersController(...) | Конструктор | - | Ініціалізація даними або XML |

Продовження таблиці 4.4

| **Назва** | **Тип** | **Розмір (байт)** | **Призначення** |
|---|---|---|---|
| ~TinySrpiteLayersController() | Деструктор | - | Очищення ресурсів |

Згодом, після проведення всіх вище описаних процедур, ігрову мапу необхідно передати контроллеру ігрової мапи.

За для створення ігрових мап створено клас TinyMap

Детальна структура класу TinyMap представлена до огляду у наступній таблиці 4.5.

Таблиця 4.5

**Змінні та методи класу TinyMap**

| **Назва** | **Тип** | **Розмір (байт)** | **Призначення** |
|---|---|---|---|
| tileMap | std::unique_ptr<TileMap> | ~8 | Вказівник на об’єкт карти |
| position | Coord | ~8 | Координати розміщення мапи |
| animationController | AnimationController | ~128 | Керує анімаціями на мапі |
| gameObjects | std::vector<std::shared_ptr<IGameObject>> | ~24 | Список ігрових об'єктів |
| gameClasses | std::vector<std::shared_ptr<IGameObject>> | ~24 | Класи об'єктів (шаблони) |

Клас TinyMap можуть наслідувати інші класи ігрових мап, за для того щоб на його основі користувачі могли створювати власні ігрові світи.

Таким чином, наслідуючи клас TinyMap, мною було створено клас ігрової мапи моєї гри WonderWorld.

У таблиці 4.6 представлено детальний огляд структури та полів класу WonderWorld.

Таблиця 4.6

**Поля класу WonderWorld**

| Назва | Тип | Розмір (байт) | Призначення |
|---|---|---|---|
| player | std::shared_ptr<Player> | ~8 | Головний персонаж гри |
| enemys | std::vector<std::shared_ptr<Pawn>> | ~24 | Список ворожих персонажів |
| audioController | AudioController | ~32 | Відповідає за звуки |
| pawnUpdate | std::mutex | ~40 | Синхронізація оновлення Pawn |
| posUpdate | std::mutex | ~40 | Синхронізація оновлення позиції |

Як ми можемо спостерігати зі структури класу WonderWorld, клас наслідує та перевизначає методи класу TinyMap, такі як Update, та OnTrigerEnter.

Така поліморфна архітектура дозволяє нам багаторазово наслідувати та перевизначати методи та змінні класу TinyMap під наші власні потреби.

За для представлення одиниці об’єкту спарйту та було створено клас Rect.

Клас Rect є одним із найголовніших класів, адже він використовується для пресдтавленя ігрових об’єктів, та виконує їх рендер. Тому, за для повного розуміння його полів та методів, необхідно розглянути його структурну таблицю 4.7 нижче.

Таблиця 4.7

**Змінні та методи класу Rect**

| Назва | Тип | Розмір (байт) | Призначення |
|---|---|---|---|
| OnMouseHover, OnMouseOver | MouseHoverHandler | ~8 | Обробники подій наведення миші |

Продовження таблиці 4.7

| Назва | Тип | Розмір (байт) | Призначення |
|---|---|---|---|
| OnMouseClick | MouseClickHandler | ~8 | Обробник події кліку миші |
| OnCollisionEnterHandler | OnCollisionEnter | ~8 | Обробник події зіткнення |
| renderInstance | std::unique_ptr<RectRenderInstance> | ~8 | Об'єкт, що відповідає за рендеринг |
| collision | std::shared_ptr<ICollision> | ~8 | Компонент колізії |

За для оптимізації роботи гри було створено контроллер шейдерів для рушія, котрий буде брати на себе задачу створення, компіляції, збереження та надання доступу до будь-якого завантаженого шейдера.

Нижче представлена огляд структури класу ShadersController (таблиця 4.8).

Таблиця 4.8

**Структура класу ShadersController**

| Назва | Тип | Розмір (байт) | Призначення |
|---|---|---|---|
| shaders | static std::unique_ptr<std::unordered_map<GLuint, std::unique_ptr<Shader>>> | ~8 | Контейнер для всіх шейдерів за ID |
| LoadShader(...) | void | - | Завантаження шейдера |
| GetShader(GLuint) | Shader* | 4 | Отримати шейдер |

Продовження таблиці 4.8

| **Назва** | **Тип** | **Розмір (байт)** | **Призначення** |
|---|---|---|---|
| SetVec2, SetVec3, SetVec4 | void | - | Передати вектори у шейдер |
| SetMat4 | void | - | Передати матрицю 4×4 |
| SetColor | void | - | Передати колір у шейдер |
| GetShaderID(string _view) | const GLuint& | 4 | Отримати ID шейдера за назвою |
| Use, Disable | void | - | Активувати або деактивувати шейдер |
| SetBool, SetInt, SetFloat | void | - | Передати змінні типу bool/int/float у шейдер |

Як ми можемо спостерігати з представленого матеріалу мого класу контролера для шейдерів, клас зберігає шейдери у розумному вказівнику shaders, котрий у свою чергу зберігає вказівку на unordered_map елемент, котрий зберігає об’єкт класу Shader за його унікальним ідентифікатором.

Також, за для швидкого доступу до будь-якого наявного шейдера у класі створено методи за для передачі параметрів до шейдера, його активації та деактивації.

За для покращення та оптимізації алгоритмів пошуку, було прийнято рішення використовувати unordered_map.

Також у класі наявні методи для завантаження шейдера, котрі приймають його унікальну назву, шлях до vertex шейдера та fragment шейдеру.

Методика використання шейдеру у моєму рушії не тільки оптимізує роботу гри, але й надасть можливість зручно та ефективно налаштовувати рендер будь-якого ігрового об’єкту під актуальні потреби.

Використання шейдеру це звісно добре, але помим них також необхідно створити класи методів рендеру, котрі будуть готувати всі необхідні дані й розрахунки, необхідні нам для рендеру того чи іншого ігрового об’єкту, враховуючи їх специфікації та особливості.

За для реалізації цієї поставленої задачі, було створено клас RenderInstance, котрий має відповідати всім посталвеним потребам, та брати на свої плечи задачу із реалізації та створення методик рендеру.

Розглянемо структуру класу RenderInstance, котра представлена до огляду нижче (таблиця 4.9)!

Таблиця 4.9

**Структура класу RenderInstance**

| **Назва** | **Тип** | **Розмір (байт)** | **Призначення** |
|---|---|---|---|
| VBO, VAO, EBO | unsigned int | 12 | Буфери для вершин, атрибутів та індексів |
| GenerateVertices() | virtual void | - | Абстрактний метод генерації вершин |
| Initialize() | virtual void | - | Ініціалізація буферів і даних |
| UpdateVertices(...) | void | - | Оновлення геометрії об’єкта |
| SetVBO/VAO/EBO, GetVBO/VAO/EBO | void, const unsigned int& | 4 | Доступ до OpenGL-буферів |
| GetVertices, SetVertices | const std::vector<float> &, void | 32 | Отримати чи задати вершини |
| Render() | virtual void | - | Метод рендерингу (чисто віртуальний) |

Клас RenderInstance реалізовано таким чином, що стратегію рендеру можна змінити в будь який момент!

Використовуючи саме такий підхід, ми можемо не прив`язуватись до одної методики рендеру, а використовувати та створювати альтернативні методики, котрі будуть реалізовувати та рендерити наші ігрові об’єкти, враховуючи усі їх тонкощі та особливості.

Також, “під капотом” ігрового рушія, вороги мають реалізований алгоритм пошуку найшвидшого шляху до вказаного об’єкту (A*).

Наш рушій має клас Pawn, котрий бере на себе всю відповідальність за керування та представлення персонажів у грі.

Розглянемо більш детально структуру класу Pawn у таблиці 4.10.

Таблиця 4.10

**Структура класу Pawn**

| **Назва** | **Тип** | **Розмір (байт)** | **Призначення** |
|---|---|---|---|
| OnMouseHover, OnMouseOver | MouseHoverHandler | 8 | Обробка подій наведення курсору |
| OnMouseClick | MouseClickHandler | 8 | Обробка кліку миші |
| OnCollisionEnterHandler | OnCollisionEnter | 8 | Обробка зіткнень |
| target | std::weak_ptr<Pawn> | ~8 | Ціль персонажа для ІІ або бою |
| renderInstance | std::unique_ptr<RectRenderInstance> | ~8 | Рендеринг |
| animations | AnimationController | залежить | Керує анімаціями персонажа |

Продовження таблиці 4.10

| **Назва** | **Тип** | **Розмір (байт)** | **Призначення** |
|---|---|---|---|
| audioController | AudioController | залежить | Звукові ефекти |
| playerImages | std::unique_ptr<SlicedImage> | ~8 | Графічне представлення персонажа |
| characterType | CharacterTypes | 4 | Тип персонажа (гравець, ворог тощо) |
| layer | Layer | 4 | Графічний шар |
| action | Actions | 4 | Поточна дія (атакується, рухається) |
| title | std::string | ~32 | Назва персонажа |
| speed, maxSpeed, minSpeed | float | 4×3 = 12 | Швидкість руху |
| health, maxHealth | float | 4×2 = 8 | Поточне та максимальне здоров'я |
| weight | float | 4 | Вага об'єкта |
| viewWidth, viewDistance | float | 4×2 = 8 | Поле зору та його дальність |
| damage, damageDistance | float | 4×2 = 8 | Сила атаки та її радіус |
| damageObject, interactiveObject | std::weak_ptr<IGameObject> | ~8×2 = 16 | Цілі для атаки або взаємодії |
| interactiveDistance | float | 4 | Радіус для взаємодії |
| isDead, isPlayable, isHidden, kinematic | bool | 1×4 = 4 | Стан персонажа |

У верхній лівій частині екрану також розміщений індикатор здоров’я гравця, котрий також може або зменшуватися (при отриманні гравцем шкоди для здоров’я), або ж відновлювати свої показники з часом.

На даному зображенні не можна знайти слідів будь-яких “артефактів”, багів або помилок допущених у процесі рендеру, з чого ми можемо зробити висновок, що створена мною система рендеру, шейдерів, та їх взаємодії - працює справно, що не може не викликати захвату!

Звичайно, мапа ігрового всесвіту не завершується цим невеликим відрізком, продемонстрованим на першому зображені.

Пройшовши трохи правіше, ми можемо спостерігати інші локації (рис. 4.3).


Рис. 4.3. Інша локація ігрової мапи WonderWorld

Як можна помітити, анімації у гравця змінюються в залежності від напрямку його руху та дії, котру він виконує в момент часу.

Також, не мало важливим моментом, котрий варто взяти до уваги є те, що гравець завжди центрується камерою по центру ігрового екрану. За для реалізації цього було створено окремий клас Camera, котрий вираховує view matrix та view projection значення, котрі потім використовуються у шейдерах ігрових об’єктів

за для того, щоб вирахувати їхню достовірну поточну позицію на ігровому екрані.

Такий прийом використання камери не тільки дозволяє оптимізувати роботу нашої гри (нема потреби перераховувати VAO, VBO та EBO значення для об’єктів кожен раз), але й реалізувати зручний та практичний інструмент для слідкування та центрування нашого гравця на екрані.

Також, за для економії ресурсів процесора та відеокарти, особлива увага була приділена ігровим об’єктам, котрі знаходяться поза межою досяжності об’єктиву камери. Було прийнято вагоме за для оптимізації гри рішеня не використовувати рендер до цих елементів, а лише прорахувати логіку поведінки тих об’єктів, котрі дійсно цього потребують.

## ВИСНОВКИ ДО РОЗДІЛУ 4

У цьому розділі було реалізовано основні частини програмного забезпечення гри, які визначають її загальну працездатність, інтерактивність і візуальну якість. Розробка включала основну логіку запуску програми, включаючи ініціалізацію ресурсів, зчитування даних і компіляцію шейдерів, а також створення початкового інтерфейсу користувача. Останній включає екран завантаження з візуальним прогресбаром, порадами для гравця та шрифти, які можна змінити.

Особлива увага була приділена розробці WonderWorld, ігрової мапи, яка демонструє якісну візуальну складову та складну внутрішню логіку, включаючи анімацію дерев, рух ворогів, взаємодію з колізіями та появу персонажів у певних точках. Усі елементи сцени мають логічну та стійку взаємодію завдяки правильній організації класів, таких як Pawn, Camera, TileMap і RenderInstance.

Реалізація алгоритму пошуку шляху (A*) для ворогів є значним досягненням, що свідчить про наявність базових елементів штучного інтелекту в проєкті. Крім того, візуальні частини гри працюють без артефактів або помилок

рендеру, що свідчить про надійність системи шейдерів, управління камерами та анімаційного контролю, які були створені.

Також, була приділена увага оптимізації рендеру та розрахунку логіки поведінки ігрових об’єктів задіяних на мапі. Було створено камеру, котра розраховує view matrix та view projection значення, необхідні подальшому для розрахування у шейдері поточної позиції ігрових об’єктів на мапі, що значно покращує працездатність та загальну оптимізація гри!

Усе це стало можливим завдяки правильній структурі рушія, продуманій архітектурі коду та чіткому розподілу відповідальності між класами. Таким чином, програмне забезпечення не тільки виконує завдання, але й створює стійку платформу для подальшого створення повноцінної RPG-гри.

# ВИСНОВКИ

У процесі виконання дипломної роботи було створено повний прототип комп’ютерної гри RPG з відкритим світом, розробленої мовою програмування C++ із використанням API OpenGL. Проєкт охопив основні елементи сучасної геймдев-розробки, включаючи створення структури рушія, дизайн візуального середовища, логіку персонажів і інтерактивні об’єкти.

Вимоги до гри, архітектура рушія та ядро, які забезпечують анімацію, логіку поведінки персонажів, зіткнення, навігацію, рендеринг і взаємодію з користувачем, були розроблені на основі аналізу існуючих RPG-проектів. Клас Pawn став основним компонентом рушія, який містить основні функції всіх живих об’єктів гри, включаючи гравця, ворогів і NPC. Система легко масштабує гру та додає нові механіки завдяки своїй модульній структурі.

Окрема увага була приділена тому, як організувати інформаційний контент. Створена структура папки Content для зберігання ігрових ресурсів, яка містить всі необхідні компоненти, такі як шрифти, зображення, мапи, скрипти, анімації тощо. Використовуються формати в грі. tmx та tsx, що дозволяє легко працювати з рівнями, об’єктами, шарами та анімаціями за допомогою редактора Tiled. Усі ці дані обробляються відповідними XML-контролерами, які були реалізовані в рушії.

Крім того, проєкт включає систему шейдерів, екран завантаження гри, кастомну камеру з функцією автоматичного центрування гравця та алгоритм A* для пошуку шляху ворогів. Графічний компонент розроблений за допомогою методів оптимізації, що гарантує стабільну роботу без артефактів і помилок візуалізації. Добре організована система класів, яка працює за принципами об’єктно-орієнтованого програмування, контролює усі взаємодії між компонентами.

Таким чином, мета була досягнута — створити прототип відкритого світу RPG-гри.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Павловська Т. А. C#. Програмування мовою високого рівня : підручник для вищих навчальних закладів. — СПб.: Пітер, 2009. — 432 с.: іл.

2. Холцнер С. Visual C++ 6: навчальний курс. — СПб.: ЗАТ «Видавництво «Пітер», 1999. — 576 с.

3. Зеленський А. С., Лисенко В. С., Баран С. В. Методичні вказівки для самостійного вивчення роботи з базами даних на Visual C++ з використанням об’єктів ActiveX Data Object (ADO) / Криворізький економічний інститут ДВНЗ «КНЕУ імені Вадима Гетьмана». — Кривий Ріг: КЕІ, 2008. — 54 с.

4. Зеленський А. С., Лисенко В. С., Баран С. В. Методичні вказівки до виконання лабораторних та індивідуальних робіт на основі прикладів розробки програмного забезпечення у Visual C++ 6.0 / Криворізький економічний інститут ДВНЗ «КНЕУ імені Вадима Гетьмана». — Кривий Ріг: КЕІ, 2007. — 63 с.

5. Грегорі К. Використання Visual C++ 6. Навчальний посібник. — СПб. – К.: Видавничий дім «Вільямс», 1999. — 864 с.

6. Болью А. Вивчаємо SQL / А. Болью. — О’Райлі Медія, 2009. Нормалізація таблиць бази даних — основа проєктування структури реляційних баз даних. Усунення надмірності та дублювання інформації [Електронний ресурс]. — Режим доступу: http://www.wwwmaster.ru/article.php?nart=21

7. Шлейфер М. OpenGL. Професійне програмування. — К.: Діалектика, 2007. — 384 с.

8. Шульц Т. Основи ігрової графіки з OpenGL. — Харків: Фоліо, 2015. — 296 с.

9. Керніган Б., Рітчі Д. Мова програмування С. Класичне видання. — К.: BHV, 2002. — 274 с.

# ДОДАТКИ

Додаток А

Класи формування ігрової мапа

**TinyXml.cpp**

```
#include "TinyXml.h"
#include "TileMap.h"
#include "../../Content/Scripts/UI/ProgressBar/ProgressBar.h"
#include "../../Content/Scripts/UI/ProgressBar/utilities.h"

std::string TinyXml::root_dir = "Content/";

std::string TinyXml::GetPathToTileSource(std::string path)
{
   std::filesystem::path fullPath = std::filesystem::absolute(path);
   std::filesystem::path fullSource = std::filesystem::path(root_dir) / path.substr(8);
   return root_dir + fullSource.string();
}

std::vector<Coord> TinyXml::ParsePolygon(std::string polygon, const Coord& offset)
{
   std::vector<Coord> coords;
   std::istringstream stream(polygon);
   std::string point;

   while (std::getline(stream, point, ' ')) {
      std::istringstream coordStream(point);
      std::string xStr, yStr;

      if (std::getline(coordStream, xStr, ',') && std::getline(coordStream, yStr)) {
         try {
            double x = std::stod(xStr) + offset.X;
            double y = std::stod(yStr) + offset.Y;
            coords.emplace_back(x, y);
         }
         catch (const std::invalid_argument& e) {
            throw std::runtime_error("Invalid coordinate format in points attribute");
         }
      }
      else {
         throw std::runtime_error("Invalid points format");
      }
   }

   return coords;
}

tinyxml2::XMLError TinyXml::ReadDoc(tinyxml2::XMLDocument& doc, std::string path)
{
   if (doc.LoadFile(path.c_str()) != tinyxml2::XML_SUCCESS) {
      std::cerr << "Failed to load map file." << std::endl;
```

```
        return tinyxml2::XML_ERROR_FILE_READ_ERROR;
    }
    return tinyxml2::XML_SUCCESS;
}

std::unique_ptr<TileMap> TinyXml::LoadMap(
    std::string path,
    std::string title,
    std::weak_ptr<ProgressBar> progressBar
)
{
    NextProgressBarValue(progressBar);

    tinyxml2::XMLDocument doc;
    if (TinyXml::ReadDoc(doc, path) != tinyxml2::XML_SUCCESS) {
        return std::make_unique<TileMap>();
    }

    //Map
    std::unique_ptr<TileMap> map = std::make_unique<TileMap>();

    tinyxml2::XMLElement* mapElement = doc.FirstChildElement("map");
    map->size.width = mapElement->IntAttribute("width");
    map->size.height = mapElement->IntAttribute("height");

    map->title = title;

    map->infinite = mapElement->BoolAttribute("infinite");

    map->nextLayerId = mapElement->IntAttribute("nextlayerid");
    map->nextObjectId = mapElement->IntAttribute("nextobjectid");

    map->orientation = mapElement->Attribute("orientation");
    map->renderOreder = mapElement->Attribute("renderorder");

    map->tileSize.height = mapElement->IntAttribute("tileheight");
    map->tileSize.width = mapElement->IntAttribute("tilewidth");

    NextProgressBarValue(progressBar);

    std::cout << "[MapLoad]::TILESETS::LOAD" << std::endl;

    //Tilesets
    TilesetsController tilesetsController(mapElement, path);

    std::cout << "[MapLoad]::TILESETS::OK" << std::endl << std::endl;

    NextProgressBarValue(progressBar);

    std::cout << "[MapLoad]::CLASSES::LOAD" << std::endl;
```

```
    //Classes
    TinyClassController objects;
    tinyxml2::XMLElement* classes = mapElement->FirstChildElement("group");

    std::cout << classes->Attribute("name") << std::endl;

    if (classes != nullptr) {
        objects = TinyClassController(classes);
    }

    std::cout << "[MapLoad]::CLASSES::OK" << std::endl << std::endl;

    NextProgressBarValue(progressBar);

    std::cout << "[MapLoad]::SPRITES::LOAD" << std::endl;
    //Sprites
    TinySrpiteLayersController layersController;
    tinyxml2::XMLElement* sprites = classes->NextSiblingElement("group");

    if (sprites != nullptr) {
        layersController = TinySrpiteLayersController(sprites);
    }

    map->tilesetsController = tilesetsController;
    map->classesController = objects;
    map->spriteLayersController = layersController;

    NextProgressBarValue(progressBar);

    return std::move(map);
}
```

## TilesetsController.cpp

```
#include "TilesetsController.h"

TilesetsController::TilesetsController()
{
    tilesets = std::vector<std::shared_ptr<Tileset>>();
}

TilesetsController::TilesetsController(std::vector<std::shared_ptr<Tileset>> tilesets)
{
    this->tilesets = tilesets;
}

TilesetsController::TilesetsController(tinyxml2::XMLElement* element, std::string path)
{
    LoadTilesets(element, tilesets, path);
}
```

```
TilesetsController::~TilesetsController()
{
}

void TilesetsController::LoadTilesets(tinyxml2::XMLElement* element,
std::vector<std::shared_ptr<Tileset>>& tilesets, std::string path)
{
   for (tinyxml2::XMLElement* tileset = element->FirstChildElement("tileset");
      tileset != nullptr;
      tileset = tileset->NextSiblingElement("tileset")
      ) {

      std::string source = tileset->Attribute("source");
      if (!source.empty()) {
         std::string pth = fs::path(path).parent_path().string() + "/";
         tilesets.push_back(
            LoadTileset(
               tileset->IntAttribute("firstgid"),
               (pth + source).c_str()
            )
         );
         std::cout << source << " OK" << std::endl;
         continue;
      }
      std::cout << source << " ERROR!" << std::endl;
   }
}

std::shared_ptr<Tileset> TilesetsController::LoadTileset(int firstgId, std::string path)
{
   tinyxml2::XMLDocument doc;
   if (TinyXml::ReadDoc(doc, path) != tinyxml2::XML_SUCCESS) {
      return nullptr;
   }

   tinyxml2::XMLElement* tilesetElement = doc.FirstChildElement("tileset");
   std::string name = tilesetElement->Attribute("name");

   int tileWidth = tilesetElement->IntAttribute("tilewidth");
   int tileHeight = tilesetElement->IntAttribute("tileheight");

   int tileCount = tilesetElement->IntAttribute("tilecount");
   int columns = tilesetElement->IntAttribute("columns");

   TilesetPropertiesController tilesetProperties = TilesetPropertiesController();

   //Load image
   tinyxml2::XMLElement* image = tilesetElement->FirstChildElement("image");
   std::string source = image->Attribute("source");
```

```
int width = image->IntAttribute("width");
int height = image->IntAttribute("height");

if (source.length() == 0) {
   return nullptr;
}

source = TinyXml::GetPathToTileSource(source);
Image image_obj = ImagesController::LoadImg(source.c_str(), name);
if (image_obj.GetImage() <= 0) {
   return nullptr;
}

std::unordered_map<int, std::shared_ptr<Tile>> tiles;

//Tiles
for (tinyxml2::XMLElement* tileset = tilesetElement->FirstChildElement("tile");
   tileset != nullptr;
   tileset = tileset->NextSiblingElement("tile")
   ) {

   int tile_id = tileset->IntAttribute("id");

   //Properties
   tinyxml2::XMLElement* properties = tileset->FirstChildElement("properties");
   if (properties != nullptr) {
      tilesetProperties = TilesetPropertiesController(properties);
   }

   //animation
   tinyxml2::XMLElement* animation = tileset->FirstChildElement("animation");

   std::shared_ptr<Animation> animation_obj = nullptr;
   if (animation != nullptr) {
      animation_obj = std::make_shared<Animation>(animation);
   }
   //Object group
   tinyxml2::XMLElement* objectGroup = tileset->FirstChildElement("objectgroup");

   if (objectGroup == nullptr) {
      std::unique_ptr<Tile> tile = std::make_unique<Tile>(
         tile_id,
         nullptr,
         animation_obj
      );

      tile->SetTilesetProperties(tilesetProperties);

      tiles[tile_id] = std::move(tile);
```

```
    continue;
}

//Object
tinyxml2::XMLElement* object = objectGroup->FirstChildElement("object");

if (object == nullptr) {
    std::unique_ptr<Tile> tile = std::make_unique<Tile>(
        tile_id,
        nullptr,
        animation_obj
    );

    tile->SetTilesetProperties(tilesetProperties);
    tiles[tile_id] = std::move(tile);
    continue;
}

int object_id = object->IntAttribute("id");
std::string object_name = "";
std::string object_type = "";

{
    const char* temp_name = object->Attribute("name");
    const char* temp_type = object->Attribute("type");

    if (temp_name != nullptr) {
        object_name = temp_name;
    }
    if (temp_type != nullptr) {
        object_type = temp_type;
    }
}

Coord object_pos = {
    object->DoubleAttribute("x"),
    object->DoubleAttribute("y")
};

Size object_size = {
    object->DoubleAttribute("width"),
    object->DoubleAttribute("height")
};

tinyxml2::XMLElement* polygon = object->FirstChildElement("polygon");

if (polygon == nullptr) {
    std::unique_ptr<Tile> tile = std::make_unique<Tile>(
        tile_id,
        std::make_shared<BoxCollision>(
```

```
                object_pos,
                object_size,
                object_id,
                object_name,
                object_type
            ),
            animation_obj
        );

        tile->SetTilesetProperties(tilesetProperties);

        tiles[tile_id] = std::move(tile);
        continue;
    }

    std::unique_ptr<Tile> tile = std::make_unique<Tile>(
        tile_id,
        std::make_shared<PoligonCollision>(
            TinyXml::ParsePolygon(polygon->Attribute("points"), object_pos),
            object_id,
            object_name,
            object_type
        ),
        animation_obj
    );

    tile->SetTilesetProperties(tilesetProperties);

    tiles[tile_id] = std::move(tile);
  }

  return std::make_shared<Tileset>(
    firstgId, tileWidth, tileHeight, width, height,
    tileCount, columns, name, source.c_str(),
    image_obj, tiles, tilesetProperties
  );
}

int TilesetsController::GetSize()
{
  return tilesets.size();
}

std::weak_ptr<Tileset> TilesetsController::GetTilesetByTileId(int tileId)
{
  for (std::shared_ptr<Tileset>& tileset : tilesets) {
    if (tileset->IsContains(tileId)) {
      return tileset;
    }
  }
```

```
    return std::make_shared<Tileset>();
}

std::weak_ptr<Tileset> TilesetsController::operator[](int index)
{
    for (int i = 0; i < tilesets.size(); i++)
    {
        if (i == index)
        {
            return tilesets[i];
        }
    }
    return std::make_shared<Tileset>();
}

std::weak_ptr<Tileset> TilesetsController::operator[](std::string name)
{
    for (int i = 0; i < tilesets.size(); i++)
    {
        if (tilesets[i]->GetName() == name)
        {
            return tilesets[i];
        }
    }
    return std::make_shared<Tileset>();
}
```

## TinyClassController.cpp

```
#include "TinyClassController.h"

TinyClassController::TinyClassController()
{
}

TinyClassController::TinyClassController(std::unordered_map<int,     std::shared_ptr<TinyClass>>
classes)
{
    this->classes = classes;
}

TinyClassController::TinyClassController(tinyxml2::XMLElement* element)
{
    LoadClasses(element, classes);
}

TinyClassController::~TinyClassController()
{
}
```

Продовження додатку А

```
void TinyClassController::LoadClasses(tinyxml2::XMLElement* element, std::unordered_map<int,
std::shared_ptr<TinyClass>>& classes)
{
   for (tinyxml2::XMLElement* child = element->FirstChildElement("group");
      child != nullptr;
      child = child->NextSiblingElement("group")
   )
   {
      std::unique_ptr<TinyClass> temp = std::make_unique<TinyClass>(child);
      classes[temp->GetId()] = std::move(temp);
   }
}

std::unordered_map<int, std::shared_ptr<TinyClass>>::iterator TinyClassController::begin()
{
   return classes.begin();
}

std::unordered_map<int, std::shared_ptr<TinyClass>>::iterator TinyClassController::end()
{
   return classes.end();
}

int TinyClassController::GetSize()
{
   return classes.size();
}

std::weak_ptr<TinyClass> TinyClassController::GetById(int id)
{
   auto it = classes.find(id);
   if (it == classes.end()) {
      return {};
   }
   return it->second;
}

std::weak_ptr<TinyClass> TinyClassController::operator[](std::string_view name)
{
   for (auto& it : classes) {
      if (it.second->GetName() == name) {
         return it.second;
      }
   }
   return {};
}

std::weak_ptr<TinyClass>TinyClassController::operator[](int index)
{
   if (index < 0 || index >= classes.size())
```

```
    {
        return {};
    }

    auto it = classes.begin();
    std::advance(it, index);
    return it->second;
}
```

## TinySrpiteLayersController.cpp

```
#include "TinySrpiteLayersController.h"

TinySrpiteLayersController::TinySrpiteLayersController()
{
}

TinySrpiteLayersController::TinySrpiteLayersController(std::unordered_map<int,
std::shared_ptr<TinySpriteLayer>> spriteLayers)
{
    this->spriteLayers = spriteLayers;
}

TinySrpiteLayersController::TinySrpiteLayersController(tinyxml2::XMLElement* element)
{
    LoadSpriteLayers(element, spriteLayers, spriteLayerOrders);
}

TinySrpiteLayersController::~TinySrpiteLayersController()
{
}

void TinySrpiteLayersController::LoadSpriteLayers(
    tinyxml2::XMLElement* element,
    std::unordered_map<int, std::shared_ptr<TinySpriteLayer>>& spriteLayers,
    std::vector<int>& spriteLayerOrders
)
{
    for (tinyxml2::XMLElement* group = element->FirstChildElement("group");
        group != nullptr;
        group = group->NextSiblingElement("group")
        ) {
        for (tinyxml2::XMLElement* layer = group->FirstChildElement("layer");
            layer != nullptr;
            layer = layer->NextSiblingElement("layer")
            ) {
            if (layer) {
                std::unique_ptr<TinySpriteLayer> temp = std::make_unique<TinySpriteLayer>(layer);
                int id = temp->GetId();  // Сохраняем ID перед move!
                spriteLayerOrders.push_back(id);
```

```
            spriteLayers[temp->GetId()] = std::move(temp);
            std::cout << layer->Attribute("name") << " OK" << std::endl;
            continue;
         }
         std::cout << layer->Attribute("name") << " ERROR!" << std::endl;
      }
   }
}

int TinySrpiteLayersController::GetSize()
{
   return spriteLayers.size();
}

std::weak_ptr<TinySpriteLayer> TinySrpiteLayersController::GetById(int id)
{
   auto it = spriteLayers.find(id);
   if (it == spriteLayers.end())
   {
      return {};
   }
   return it->second;
}

std::unordered_map<int, std::shared_ptr<TinySpriteLayer>>::iterator
TinySrpiteLayersController::begin()
{
   return spriteLayers.begin();
}

std::unordered_map<int, std::shared_ptr<TinySpriteLayer>>::iterator
TinySrpiteLayersController::end()
{
   return spriteLayers.end();
}

const std::vector<int>& TinySrpiteLayersController::GetSpriteLayerOrders()
{
   return spriteLayerOrders;
}

std::weak_ptr<TinySpriteLayer> TinySrpiteLayersController::operator[](int index)
{
   if (index < 0 || index >= spriteLayers.size())
   {
      return {};
   }

   auto it = spriteLayers.begin();
   std::advance(it, index);
```

```
    return it->second;
}

std::weak_ptr<TinySpriteLayer> TinySrpiteLayersController::operator[](std::string name)
{
    for (auto& item : spriteLayers)
    {
        if (item.second->GetName() == name)
        {
            return item.second;
        }
    }
    return {};
}
```

**TinyMap.cpp**

```
    #include "TinyMap.h"
#include "../GameStatuses.h"
#include "../animator/VertexAnimation.h"

void TinyMap::MoveCollison(std::shared_ptr<ICollision> collision, Coord* position)
{
    if (collision == nullptr) {
        return;
    }

    if (position == nullptr) {
        position = new Coord(this->position);
    }

    std::vector<Coord> points;

    for (Coord& coord : collision->GetPoints()) {
        coord += *position;
        points.push_back(coord);
    }

    collision->SetPoints(points);
}

void TinyMap::Initialize()
{
    //Math Map Texture Position
    for (const int& spriteLayerId : tileMap->spriteLayersController.GetSpriteLayerOrders()) {
        std::shared_ptr<TinySpriteLayer> spriteLayer = tileMap-
>spriteLayersController.GetById(spriteLayerId).lock();
        if (!spriteLayer) {
            continue;
        }

        const Size spriteLayerSize = spriteLayer->GetSize();
```

```
std::cout << "Layer: " << spriteLayer->GetName() << std::endl;

for (int chunk_id = 0; chunk_id < spriteLayer->GetChunksCount(); chunk_id++) {
   std::weak_ptr<TinyChunk> weakChunk = spriteLayer->operator[](chunk_id);
   std::shared_ptr<TinyChunk> chunk = weakChunk.lock();

   if (chunk == nullptr) {
      continue;
   }

   for (int i = 0; i < chunk->size.height; i++) {
      for (int j = 0; j < chunk->size.width; j++) {
         int tileId = chunk->tileIds[i][j];

         if (!tileId) {
            continue;
         }

         std::shared_ptr<Tileset> tileset = tileMap->tilesetsController.GetTilesetByTileId(tileId).lock();
         if (tileset == nullptr) {
            continue;
         }

         std::shared_ptr<Tile> tile = tileset->GetTileById(tileId).lock();

         std::shared_ptr<ICollision> collision = tile ? tile->GetCollision().lock() : nullptr;
         std::shared_ptr<Animation> tileAnimation = tile ? tile->GetAnimation().lock() : nullptr;

         std::shared_ptr<Rect> rect = nullptr;
         std::unique_ptr<VertexAnimation> animation = nullptr;

         {
            int ciclesCount = tile == nullptr
               ? 0
               : tileAnimation != nullptr
               ? tileAnimation->FrameCount()
               : 0;

            Coord vertex1, vertex2;
            Coord textureVertex1, textureVertex2;

            std::string random_str = GenerateRandomString(5);
            std::string obj_title = random_str.c_str();

            for (int k = 0; k < (ciclesCount ? ciclesCount : 1); k++) {

               const int tileWidth = tileset->GetTileWidth();
               const int tileHeight = tileset->GetTileHeight();
```

```
                const int width = tileset->GetWidth();
                const int height = tileset->GetHeight();

                const int columns = tileset->GetColumns();

                int tileValue = tileId > tileset->GetFirstgId()
                   ? tileId - tileset->GetFirstgId()
                   : tileId;

                int atlasX = tileValue % columns;
                int atlasY = tileValue / columns;

                float tileU = (float)tileWidth / (float)width;
                float tileV = (float)tileHeight / (float)height;

                if (rect == nullptr) {
                   const int chunk_j = j + chunk->coord.X;
                   const int chunk_i = i + chunk->coord.Y;

                   vertex1 = Coord(Window::PixelToGLX(((chunk_j + 1) * tileWidth) + position.X), Window::PixelToGLY(((chunk_i + 1) * tileHeight) + position.Y));
                   vertex2 = Coord(Window::PixelToGLX(((chunk_j * tileWidth)) + position.X), Window::PixelToGLY((chunk_i * tileHeight) + position.Y));
                }

                textureVertex1 = Coord(((float)atlasX + 1.0f) * tileU, 1.0f - ((float)(atlasY + 1) * (float)tileV));
                textureVertex2 = Coord((float)atlasX * (float)tileU, 1.0f - ((float)atlasY * tileV));

                if (animation == nullptr && ciclesCount > 0) {
                   animation = std::make_unique<VertexAnimation>(
                      obj_title,
                      4,
                      true,
                      false,
                      rect.get()
                   );

                   tileId = tileAnimation->operator[](k)->tileId;
                   animation->AddFrame(tileAnimation->operator[](k)->duration, {
                      textureVertex1,
                      textureVertex2
                      });

                   tileId = k + 1 < ciclesCount
                      ? tileAnimation->operator[](k + 1)->tileId
                      : tileId;
                }
```

Продовження додатку А

```
                if (rect != nullptr) {
                    animation->AddFrame(tileAnimation->operator[](k)->duration, {
                        textureVertex1,
                        textureVertex2
                        });

                    tileId = k + 1 < ciclesCount
                        ? tileAnimation->operator[](k + 1)->tileId
                        : tileId;
                    continue;
                }

                rect = std::make_shared<Rect>(
                    obj_title,
                    vertex1,
                    vertex2,
                    textureVertex1,
                    textureVertex2
                );
            }
        }

        if (animation != nullptr) {
            animation->SetGameObject(rect.get());
            animationController.AddAnimation(std::move(animation));
        }

        Coord rectPos = rect->GetPos();
        MoveCollison(collision, &rectPos);

        if (std::shared_ptr<BoxCollision> boxCollision = std::dynamic_pointer_cast<BoxCollision>(collision)) {
            boxCollision->SetSize(boxCollision->GetSize() * Size(.7f, .7f));
            rect->SetCollision(std::make_unique<BoxCollision>(*boxCollision));
        }

        rect->GetMaterial().lock()->SetDiffuseMap(std::make_shared<Image>(tileset->GetImage()));

        if (collision) {
            rect->HookOnCollisionEnter([](IGameObject* object, IGameObject* sender, GLFWwindow* window) {

            });
        }

        gameObjects.push_back(
            rect
        );
    }
```

```
            }
        }

        std::cout << std::endl;
    }

    //Classes
    for (auto& classes : tileMap->classesController) {
        std::cout << "Class: " << classes.second->GetName() << std::endl;
        std::cout << "Size: " << classes.second->GetSize() << std::endl;

        for (std::weak_ptr<ICollision> item : *classes.second) {
            std::shared_ptr<ICollision> collision = item.lock();
            if (collision == nullptr) {
                continue;
            }

            std::cout << "Collision: " << collision->GetRootTitle() << std::endl;

            MoveCollison(collision);
            std::ostringstream oss;
            oss << collision->GetRootId() << "_" << collision->GetType();

            std::shared_ptr<Rect> circle = std::make_shared<Rect>(
                oss.str(),
                collision->GetPoints()[0],
                Size(35, 35),
                Color(1, 1, 1)
            );

            if (std::shared_ptr<BoxCollision> boxCollision =
std::dynamic_pointer_cast<BoxCollision>(collision)) {
                circle->SetCollision(boxCollision);
            }

            //circle->SetCollision(collision);
            circle->HookOnCollisionEnter([](IGameObject* object, IGameObject* sender,
GLFWwindow* window) {
                std::cout << object->GetTitle() << " collided with " << sender->GetTitle() << std::endl;
            });

            gameClasses.push_back(circle);
        }
    }
}

TinyMap::TinyMap(std::unique_ptr<TileMap> tileMap, Coord position)
{
    this->tileMap = std::move(tileMap);
    this->position = position;
```

```
    Initialize();
}

std::vector<std::weak_ptr<IGameObject>> TinyMap::GetClassesByType(std::string type)
{
    std::vector<std::weak_ptr<IGameObject>> result = std::vector<std::weak_ptr<IGameObject>>();

    for (std::shared_ptr<IGameObject>& object : gameClasses) {
        if (object->GetCollision().lock()->GetType() == type) {
            result.push_back(object);
        }
    }

    return result;
}

std::vector<std::weak_ptr<IGameObject>> TinyMap::GetClassesByName(std::string name)
{
    std::vector<std::weak_ptr<IGameObject>> classes;

    for (std::shared_ptr<IGameObject>& object : gameClasses) {
        if (object->GetCollision().lock()->GetRootTitle() == name) {
            classes.push_back(object);
        }
    }

    return classes;
}

std::weak_ptr<IGameObject> TinyMap::GetClassByName(std::string name)
{
    for (std::shared_ptr<IGameObject>& object : gameClasses) {
        if (object->GetCollision().lock()->GetRootTitle() == name) {
            return object;
        }
    }

    return std::weak_ptr<Rect>();
}

const std::vector<std::shared_ptr<IGameObject>>& TinyMap::GetGameObjects()
{
    return gameObjects;
}

const std::vector<std::shared_ptr<IGameObject>>& TinyMap::GetGameClasses()
{
    return gameClasses;
}
```

```
void TinyMap::InitializeRender()
{
   for (std::shared_ptr<IGameObject>& object : gameObjects) {
      object->InitializeRender();
   }
}

TileMap* TinyMap::GetTileMap()
{
   return tileMap.get();
}
```

**WonderWold.cpp**

```
#include "WonderWold.h"

#include "../../Characters/Enemys/Skeleton.h"
#include "../../../../Dodge/raycast/Raycast.h"
#include "../../../../Dodge/GameObjects.h"
#include "../../../../Dodge/shaders/ShadersController.h"


void WonderWold::SpawnPlayer()
{
   std::unique_ptr<Material> playerMaterial = std::make_unique<BaseFigureMaterial>();
   std::weak_ptr<IGameObject> weakPlayerSpawn = GetClassByName("PlayerSpawnPoint");
   std::shared_ptr<IGameObject> playerSpawn = weakPlayerSpawn.lock();

   if (playerSpawn == nullptr) {
      return;
   }

   playerMaterial->SetShader(
      ShadersController::GetShaderID("BaseFigure")
   );

   playerMaterial->SetDiffuse(Color(1, 1, 1));
   playerMaterial->SetDiffuseMap(
      std::make_shared<Image>(
         ImagesController::LoadImg(
            "Content/Assets/lpc_entry/males/player.png",
            "Player"
         )
      )
   );

   //playerMaterial->SetCamera(camera);

   player = std::make_shared<Player>(
      "Player",
```

```
        std::make_unique<BoxCollision>(
            playerSpawn->GetPos(),
            Size(24, 16),
            -1,
            (char*)"Player",
            (char*)"Player"
        ),
        std::move(playerMaterial),
        Directions::DOWN,
        playerSpawn->GetPos(),
        Size(64, 64),
        50,
        53,
        48,
        100,
        100,
        true,
        false,
        false
    );

    Window::GetCamera().lock()->SetObservedObj(player);

    WindowPointerController::SetPointer(
        WindowPointer<Player>("player", player)
    );

    GameObjects::Add(std::dynamic_pointer_cast<class Pawn>(player));
}

void WonderWold::SpawnSkeleton(Coord position)
{
    for (std::weak_ptr<IGameObject>& weakSpawn : GetClassesByName("SkeletonSpawnPoint")) {
        std::shared_ptr<IGameObject> spawn = weakSpawn.lock();
        if (spawn == nullptr) {
            continue;
        }

        //skeleton->SetTarget(player);
        std::unique_ptr<Material> skeletonMaterial = std::make_unique<BaseFigureMaterial>();
        skeletonMaterial->SetShader(
            ShadersController::GetShaderID("BaseFigure")
        );

        skeletonMaterial->SetDiffuse(Color(1, 1, 1));
        skeletonMaterial->SetDiffuseMap(
            std::make_shared<Image>(
                ImagesController::LoadImg(
                    "Content/Assets/lpc_entry/males/skeleton.png",
                    "Skeleton"
```

```
            )
        )
    );

    std::shared_ptr<class Skeleton> skeleton = std::make_unique<class Skeleton>(
        "Skeleton",
        std::make_unique<BoxCollision>(
            spawn->GetPos(),
            Size(24, 16),
            -1,
            (char*)"Skeleton",
            (char*)"Enemy"
        ),
        std::move(skeletonMaterial),
        Directions::DOWN,
        spawn->GetPos(),
        Size(64, 64),
        50,
        53,
        48,
        100,
        100,
        true,
        false,
        false
    );

    GameObjects::Add(std::dynamic_pointer_cast<class Pawn>(skeleton));
    enemys.push_back(std::move(skeleton));
    return;
  }
}

void WonderWold::OnTriggerEnter(IGameObject* object, IGameObject* triggeredObject)
{
}

Propertie* WonderWold::GetObjectPropertie(std::string propertie_name, IGameObject* object)
{
  return nullptr;
}

void WonderWold::Initialize()
{
  SpawnPlayer();
  SpawnSkeleton(Coord(450, 300));

  for (std::shared_ptr<IGameObject>& obj : gameObjects) {
    if (!obj->GetMaterial().lock() || obj->GetMaterial().expired()) {
      continue;
```

```
        }
    }

    for (std::shared_ptr<IGameObject>& classObj : gameClasses) {
        if (!classObj->GetMaterial().lock() || classObj->GetMaterial().expired()) {
            continue;
        }
    }

    std::unique_ptr<Audio> main = std::make_unique<Audio>("main",
"Content/Sounds/WonderWorld/main.wav");
    main->SetVolume(0.05f);

    std::unique_ptr<Audio> wind = std::make_unique<Audio>("wind",
"Content/Sounds/WonderWorld/light-wind.wav");
    wind->SetVolume(0.15f);

    audioController.Load(std::move(main));
    audioController.Load(std::move(wind));

    audioController.Play("main", true);
    audioController.Play("wind", true);
}

WonderWold::WonderWold(std::unique_ptr<TileMap>  tileMap, Coord position)
    : TinyMap(std::move(tileMap), position)
{
    Initialize();
}

void WonderWold::Update()
{
    if (!Window::GetCamera().lock() || Window::GetCamera().expired()) {
        std::cout << "NULL CAMERA!" << std::endl;
        return;
    }

    std::shared_ptr<IGameObject> lockedObserved = Window::GetCamera().lock()-
>GetObservedObj().lock();
    if (!lockedObserved) {
        std::cout << "Camera need observed object!" << std::endl;
        return;
    }

    bool isPause = Window::GetGameStatus() == GameStatuses::Pause;

    for (std::shared_ptr<IGameObject>& obj : gameObjects)
    {
        if (!isPause) {
            animationController.Play(obj->GetTitle());
```

Продовження додатку А

```
        }

        if (lockedObserved == nullptr) {
            continue;
        }

        const Coord& distance = lockedObserved->GetDistanceTo(*obj);

        if (distance.X >= Window::GetRenderResolutionView().width * 0.5f || distance.X <= -
(Window::GetRenderResolutionView().width * 0.65f) ||
            distance.Y >= Window::GetRenderResolutionView().height * 0.5f || distance.Y <= -
(Window::GetRenderResolutionView().height * 0.65f)) {
            continue;
        }

        obj->Update();
    }

    UpdatePawns();
}

void WonderWold::UpdatePawns()
{
    for (std::shared_ptr<class Pawn>& pawn : enemys)
    {
        pawn->Update();
    }

    player->Update();

    if (Window::GetCamera().lock() && !Window::GetCamera().expired()) {
        Window::GetCamera().lock()->Update();
    }
}
```

# Додаток Б

## Класи рендеру

### Rect.cpp

```
#include "Rect.h"
#include "../shaders/ShadersController.h"

std::vector<unsigned int> defaultIndicies = {
   0, 1, 2,
   2, 3, 0
};

bool Rect::MouseInRect(Mouse& mouse)
{
   float normMouseX = (mouse.GetMouseCoord().X / Window::GetRenderResolution().GetWidth()) * 2.0f - 1.0f;
   float normMouseY = 1.0f - (mouse.GetMouseCoord().Y / Window::GetRenderResolution().GetHeight()) * 2.0f;

   const Coord& vertex1 = renderInstance->GetVertex1();
   const Coord& vertex2 = renderInstance->GetVertex2();

   return (normMouseX >= vertex1.X && normMouseX <= vertex2.X &&
      normMouseY >= vertex1.Y && normMouseY <= vertex2.Y);
}

void Rect::MathPos(Coord& vertex1, Coord& vertex2)
{
   renderInstance->SetVertexes(vertex1, vertex2);

   float width = (vertex2.X - vertex1.X) * (Window::GetRenderResolution().GetWidth() / 2.0f);
   float height = (vertex2.Y - vertex1.Y) * (Window::GetRenderResolution().GetHeight() / 2.0f);

   size = Size(round(std::abs(width)), round(std::abs(height)));
   float centerX_GL = (vertex1.X + vertex2.X) / 2.0f;
   float centerY_GL = (vertex1.Y + vertex2.Y) / 2.0f;
   position = Coord(
      ((centerX_GL + 1.0f) / 2.0f) * (float)Window::GetRenderResolution().GetWidth(),
      ((1.0f - (centerY_GL + 1.0f) / 2.0f) * (float)Window::GetRenderResolution().GetHeight())
   );
}

void Rect::MathPos(Coord& position)
{
   this->position = position;
   float glCenterX = Window::PixelToGLX(position.X);
   float glCenterY = Window::PixelToGLY(position.Y);

   float glWidth = (float)size.GetWidth() / (float)Window::GetRenderResolution().GetWidth() * 2.0f;
```

```
    float glHeight = (float)size.GetHeight() / (float)Window::GetRenderResolution().GetHeight() *
2.0f;

    // Calculate the vertex1 and vertex2 coordinates
    //vertex2.X = glCenterX - glWidth / 2.0f;
    //vertex1.Y = glCenterY - glHeight / 2.0f;
    //vertex1.X = glCenterX + glWidth / 2.0f;
    //vertex2.Y = glCenterY + glHeight / 2.0f;

    renderInstance->SetVertexes(
        Coord(glCenterX + glWidth / 2.0f, glCenterY - glHeight / 2.0f),
        Coord(glCenterX - glWidth / 2.0f, glCenterY + glHeight / 2.0f)
    );
}

void Rect::MathSize(Size& size)
{
    this->size = size;
    MathPos(position);
}

void Rect::MathSide(double& sideSize, bool isWidth)
{


}

Rect::Rect()
{
    title = (char*)"Undefined";

    collision = nullptr;

    OnMouseHover = OnMouseOver = nullptr;
    OnCollisionEnterHandler = nullptr;
    OnMouseClick = nullptr;

    kinematic = false;
    this->layer = Layer::Undefined;

    moveDirection = Directions::DOWN;

    renderInstance = std::make_unique<RectRenderInstance>();
}

Rect::Rect(
    std::string title, Coord
    position, Size size, Color color,
    Directions moveDirection
)
```

Продовження додатку Б

```
{
    this->title = title;

    this->size = size;
    this->color = baseColor = color;

    OnMouseHover = OnMouseOver = nullptr;
    OnCollisionEnterHandler = nullptr;
    OnMouseClick = nullptr;

    std::unique_ptr<Material> material = std::make_unique<BaseFigureMaterial>();
    material->SetShader(
        ShadersController::GetShaderID("BaseFigure")
    );

    material->SetDiffuse(color);
    material->SetDiffuseMap(ImagesController::GetDafaultImage().lock());

    this->moveDirection = moveDirection;

    kinematic = false;
    this->layer = Layer::GameObject;

    collision = nullptr;

    renderInstance = std::make_unique<RectRenderInstance>(std::move(material));

    MathPos(position);
}

Rect::Rect(
    std::string title,
    Coord vertex1, Coord vertex2,
    Color color, Directions moveDirection
)
{
    this->title = title;

    this->color = baseColor = color;

    OnMouseHover = OnMouseOver = nullptr;
    OnCollisionEnterHandler = nullptr;
    OnMouseClick = nullptr;

    std::unique_ptr<Material> material = std::make_unique<BaseFigureMaterial>();
    material->SetShader(
        ShadersController::GetShaderID("BaseFigure")
    );

    this->moveDirection = moveDirection;
```

```
    kinematic = false;
    this->layer = Layer::GameObject;

    collision = nullptr;

    renderInstance = std::make_unique<RectRenderInstance>(std::move(material));

    MathPos(vertex1, vertex2);
}

Rect::Rect(
    std::string title,
    Coord vertex1, Coord vertex2,
    Coord textureVertex1, Coord textureVertex2,
    Color color, Directions moveDirection
)
{
    this->title = title;

    this->color = baseColor = color;

    OnMouseHover = OnMouseOver = nullptr;
    OnCollisionEnterHandler = nullptr;
    OnMouseClick = nullptr;

    std::unique_ptr<Material> material = std::make_unique<BaseFigureMaterial>();
    material->SetShader(
        ShadersController::GetShaderID("BaseFigure")
    );

    std::vector<Coord> verticies = std::vector<Coord>{
        textureVertex1, textureVertex2
    };
    material->SetDiffuseMapVerticies(verticies);

    this->moveDirection = moveDirection;

    kinematic = false;
    this->layer = Layer::GameObject;

    collision = nullptr;

    renderInstance = std::make_unique<RectRenderInstance>(std::move(material));

    MathPos(vertex1, vertex2);
}

void Rect::Update()
{
```

```
    Draw();
}

void Rect::Draw()
{
    renderInstance->Render();
}

void Rect::UpdateVertices()
{
    renderInstance->UpdateVertices();
}

void Rect::UpdateVertices(std::vector<float>& vertices)
{
    renderInstance->UpdateVertices(vertices);
}

void Rect::InitializeRender()
{
    renderInstance->Initialize();
}

std::vector<float> Rect::GetVerticesByDirection(Rect& rect, Directions moveDirection, bool returnTexCoords)
{
    Coord vertex1 = rect.renderInstance->GetVertex1();
    Coord vertex2 = rect.renderInstance->GetVertex2();

    std::weak_ptr<Material> weakMaterial = rect.renderInstance->GetMaterial();
    if (weakMaterial.expired() || !weakMaterial.lock()) {
        return rect.renderInstance->GetVertices();
    }

    std::shared_ptr<Material> material = weakMaterial.lock();

    const bool isHasDiffuseVertexs =
        material->GetDiffuseMapVerticies().size() >= 2 &&
        material->GetDiffuseMap().lock() != nullptr;
    const Coord& textCoord1 = isHasDiffuseVertexs ? material->GetDiffuseMapVerticies()[0] : Coord(0, 0);
    const Coord& textCoord2 = isHasDiffuseVertexs ? material->GetDiffuseMapVerticies()[1] : Coord(1, 1);

    if (moveDirection == Directions::UP) {
        return returnTexCoords ? std::vector<float> {
            (float)vertex2.X, (float)vertex2.Y, 0.0f, (float)textCoord1.X, (float)textCoord1.Y,
                (float)vertex1.X, (float)vertex2.Y, 0.0f, (float)textCoord2.X, (float)textCoord1.Y,
                (float)vertex1.X, (float)vertex1.Y, 0.0f, (float)textCoord2.X, (float)textCoord2.Y,
                (float)vertex2.X, (float)vertex1.Y, 0.0f, (float)textCoord1.X, (float)textCoord2.Y
```

```
    }
    : std::vector<float>{
        (float)vertex2.X, (float)vertex2.Y, 0.0f,
            (float)vertex1.X, (float)vertex2.Y, 0.0f,
            (float)vertex1.X, (float)vertex1.Y, 0.0f,
            (float)vertex2.X, (float)vertex1.Y, 0.0f
    };
}

if (moveDirection == Directions::DOWN) {
    return returnTexCoords ? std::vector<float> {
        (float)vertex1.X, (float)vertex1.Y, 0.0f, (float)textCoord1.X, (float)textCoord1.Y,
            (float)vertex2.X, (float)vertex1.Y, 0.0f, (float)textCoord2.X, (float)textCoord1.Y,
            (float)vertex2.X, (float)vertex2.Y, 0.0f, (float)textCoord2.X, (float)textCoord2.Y,
            (float)vertex1.X, (float)vertex2.Y, 0.0f, (float)textCoord1.X, (float)textCoord2.Y
    }
    : std::vector<float>{
        (float)vertex1.X, (float)vertex1.Y, 0.0f,
            (float)vertex2.X, (float)vertex1.Y, 0.0f,
            (float)vertex2.X, (float)vertex2.Y, 0.0f,
            (float)vertex1.X, (float)vertex2.Y, 0.0f
    };
}

if (moveDirection == Directions::LEFT) {
    return returnTexCoords
            ? std::vector<float> {
            (float)vertex1.X, (float)vertex1.Y, 0.0f, (float)textCoord1.X, (float)textCoord1.Y,
            (float)vertex1.X, (float)vertex2.Y, 0.0f, (float)textCoord2.X, (float)textCoord1.Y,
            (float)vertex2.X, (float)vertex2.Y, 0.0f, (float)textCoord2.X, (float)textCoord2.Y,
            (float)vertex2.X, (float)vertex1.Y, 0.0f, (float)textCoord1.X, (float)textCoord2.Y
        }
    :
        std::vector<float>{
            (float)vertex1.X, (float)vertex1.Y, 0.0f,
            (float)vertex1.X, (float)vertex2.Y, 0.0f,
            (float)vertex2.X, (float)vertex2.Y, 0.0f,
            (float)vertex2.X, (float)vertex1.Y, 0.0f,
    };
}

if (moveDirection == Directions::RIGHT) {
    return returnTexCoords ? std::vector<float> {
        (float)vertex2.X, (float)vertex2.Y, 0.0f, (float)textCoord1.X, (float)textCoord1.Y,
            (float)vertex2.X, (float)vertex1.Y, 0.0f, (float)textCoord2.X, (float)textCoord1.Y,
            (float)vertex1.X, (float)vertex1.Y, 0.0f, (float)textCoord2.X, (float)textCoord2.Y,
            (float)vertex1.X, (float)vertex2.Y, 0.0f, (float)textCoord1.X, (float)textCoord2.Y
    }
    : std::vector<float>{
        (float)vertex2.X, (float)vertex2.Y, 0.0f,
```

```
            (float)vertex2.X, (float)vertex1.Y, 0.0f,
            (float)vertex1.X, (float)vertex1.Y, 0.0f,
            (float)vertex1.X, (float)vertex2.Y, 0.0f,
        };
    }
}

void Rect::RotateToDirection(Directions direction)
{

    Coord vertex1 = renderInstance->GetVertex1();
    Coord vertex2 = renderInstance->GetVertex2();
    if ((moveDirection == Directions::RIGHT && direction == Directions::LEFT) ||
        (moveDirection == Directions::LEFT && direction == Directions::RIGHT) ||
        (moveDirection == Directions::UP && direction == Directions::DOWN) ||
        (moveDirection == Directions::DOWN && direction == Directions::UP)) {
        std::swap(vertex1.X, vertex2.X);
        std::swap(vertex1.Y, vertex2.Y);
    }
    else if ((moveDirection == Directions::RIGHT && direction == Directions::UP) ||
        (moveDirection == Directions::UP && direction == Directions::LEFT) ||
        (moveDirection == Directions::LEFT && direction == Directions::DOWN) ||
        (moveDirection == Directions::DOWN && direction == Directions::RIGHT)) {
        float centerX = (vertex1.X + vertex2.X) / 2;
        float centerY = (vertex1.Y + vertex2.Y) / 2;

        float tempX = vertex1.X - centerX;
        float tempY = vertex1.Y - centerY;

        vertex1.X = -tempY + centerX;
        vertex1.Y = tempX + centerY;

        tempX = vertex2.X - centerX;
        tempY = vertex2.Y - centerY;

        vertex2.X = -tempY + centerX;
        vertex2.Y = tempX + centerY;
    }
    else if ((moveDirection == Directions::RIGHT && direction == Directions::DOWN) ||
        (moveDirection == Directions::DOWN && direction == Directions::LEFT) ||
        (moveDirection == Directions::LEFT && direction == Directions::UP) ||
        (moveDirection == Directions::UP && direction == Directions::RIGHT)) {
        float centerX = (vertex1.X + vertex2.X) / 2;
        float centerY = (vertex1.Y + vertex2.Y) / 2;

        float tempX = vertex1.X - centerX;
        float tempY = vertex1.Y - centerY;

        vertex1.X = tempY + centerX;
        vertex1.Y = -tempX + centerY;
```

```
        tempX = vertex2.X - centerX;
        tempY = vertex2.Y - centerY;

        vertex2.X = tempY + centerX;
        vertex2.Y = -tempX + centerY;
    }

    renderInstance->SetVertexes(vertex1, vertex2);
    moveDirection = direction;
}

const Coord& Rect::GetPos()
{
    return position;
}

const Coord& Rect::GetOpenGlPos()
{
    return Coord(Window::PixelToGLX(position.X), Window::PixelToGLY(position.Y));
}

void Rect::SetSize(Size size, bool render)
{
    MathSize(size);
    if (!render) {
        return;
    }

    UpdateVertices();
}

Size Rect::GetSize()
{
    return size;
}

void Rect::SetSideSize(Sides sides, bool render)
{
    const Coord& vertex1 = renderInstance->GetVertex1();
    const Coord& vertex2 = renderInstance->GetVertex2();

    if (sides.bottom != 0) {
        float glDelta = abs((float)sides.bottom / (float)Window::GetRenderResolution().GetWidth() *
2.0f);

        if (sides.bottom > 0) {
            renderInstance->SetVertexes(
                Coord(vertex1.X, vertex1.Y - glDelta),
                vertex2
```

```
            );
        }
        else {
            renderInstance->SetVertexes(
                Coord(vertex1.X, vertex1.Y + glDelta),
                vertex2
            );
        }
    }

    if (sides.top != 0) {
        float glDelta = abs((float)sides.top / (float)Window::GetRenderResolution().GetWidth() * 2.0f);

        if (sides.top > 0) {
            renderInstance->SetVertex2(
                Coord(vertex2.X, vertex2.Y + glDelta)
            );
        }
        else {
            renderInstance->SetVertex2(
                Coord(vertex2.X, vertex2.Y - glDelta)
            );
        }
    }

    if (sides.left != 0) {
        float glDelta = abs((float)sides.left / (float)Window::GetRenderResolution().GetWidth() * 2.0f);

        if (sides.left > 0) {
            renderInstance->SetVertex2(
                Coord(vertex2.X - glDelta, vertex2.Y)
            );
        }
        else {
            renderInstance->SetVertex2(
                Coord(vertex2.X + glDelta, vertex2.Y)
            );
        }
    }

    if (sides.right != 0) {
        float glDelta = abs((float)sides.right / (float)Window::GetRenderResolution().GetWidth() *
2.0f);

        if (sides.right > 0) {
            renderInstance->SetVertex1(
                Coord(vertex1.X + glDelta, vertex1.Y)
            );
        }
        else {
```

```
            renderInstance->SetVertex1(
                Coord(vertex1.X - glDelta, vertex1.Y)
            );
        }
    }

    size.SetWidth((vertex1.X - vertex2.X) * Window::GetRenderResolution().GetWidth() / 2.0f);
    size.SetHeight((vertex1.Y - vertex2.Y) * Window::GetRenderResolution().GetHeight() / 2.0f);

    position.X = Window::GLXToPixel((vertex1.X + vertex2.X) / 2.0f);
    position.Y = Window::GLYToPixel((vertex1.Y + vertex2.Y) / 2.0f);

    if (!render) {
        return;
    }

    UpdateVertices();
}

bool Rect::MouseHover(Mouse& mouse)
{
    if (mouse.IsEqual()) {
        return false;
    }

    const bool isHover = MouseInRect(mouse);

    if (isHover && OnMouseHover) {
        OnMouseHover(this, Window::GetWindow());
    }

    if (!isHover && OnMouseOver) {
        OnMouseOver(this, Window::GetWindow());
    }

    return isHover;
}

bool Rect::MouseClick(Mouse& mouse)
{
    if (!mouse.isClick() || !MouseInRect(mouse)) {
        return false;
    }

    if (OnMouseClick) {
        OnMouseClick(this, Window::GetWindow());
    }
    return true;
}
```

Продовження додатку Б

```
bool Rect::CollisionEnter(IGameObject& gameObject)
{
   if (!collision || collision == nullptr) {
      return false;
   }

   bool isEnter = collision->IsCollisionEnter(&gameObject);

   if (isEnter && OnCollisionEnterHandler) {
      OnCollisionEnterHandler(this, &gameObject, Window::GetWindow());
   }
}

void Rect::SetColor(Color color)
{
   this->color = color;
}

Color Rect::GetColor()
{
   return color;
}

Color Rect::GetBaseColor()
{
   return baseColor;
}

void Rect::SetPos(std::vector<Coord> vertices, bool render)
{
   MathPos(vertices[0], vertices[1]);
   if (!render) {
      return;
   }

   UpdateVertices();
}

void Rect::SetPos(Coord position, bool render)
{
   MathPos(position);
   if (!render) {
      return;
   }

   UpdateVertices();
}

void Rect::SetMaterial(std::shared_ptr<Material> material)
{
```

```
    renderInstance->SetMaterial(material);
}

std::weak_ptr<Material> Rect::GetMaterial()
{
    return renderInstance->GetMaterial();
}

void Rect::SetCollision(std::shared_ptr<ICollision> collision)
{
    this->collision = collision;
}

std::weak_ptr<ICollision> Rect::GetCollision()
{
    return collision;
}

const std::string& Rect::GetTitleString()
{
    return title;
}

std::string_view Rect::GetTitle()
{
    return title;
}

void Rect::SetTitle(std::string title)
{
    this->title = title;
}

void Rect::SetMoveDirection(Directions moveDirection)
{
    this->moveDirection = moveDirection;
}

Directions Rect::GetMoveDirection()
{
    return moveDirection;
}

const bool Rect::IsMouseOverlap()
{
    return MouseInRect(*Window::GetMouse().lock());
}

void Rect::SetLayer(Layer layer)
{
```

```
    this->layer = layer;
}

Layer Rect::GetLayer()
{
    return layer;
}

bool Rect::IsKinematic()
{
    return kinematic;
}

void Rect::SetKinematic(bool kinematic)
{
    this->kinematic = kinematic;
}

const Coord& Rect::GetDistanceTo(IGameObject& gameObject)
{
    Coord temp = gameObject.GetPos();
    return temp - position;
}

void Rect::HookMouseHover(MouseHoverHandler handler)
{
    OnMouseHover = handler;
}

void Rect::HookMouseOver(MouseHoverHandler handler)
{
    OnMouseOver = handler;
}

void Rect::HookMouseClick(MouseClickHandler handler)
{
    OnMouseClick = handler;
}

void Rect::HookOnCollisionEnter(OnCollisionEnter handler)
{
    this->OnCollisionEnterHandler = handler;
}

bool Rect::operator==(const Rect& other) const
{
    return position == other.position &&
        *renderInstance == *other.renderInstance && size == other.size &&
        color == other.color && baseColor == other.baseColor &&
        title == other.title;
```

```
}

bool Rect::operator!=(const Rect& other) const
{
   return !(*this == other);
}

Rect& Rect::operator=(const Rect&& other)
{
   if (this == &other) {
      return *this;
   }

   this->position = other.position;

   *this->renderInstance == *other.renderInstance;

   this->size = other.size;

   this->color = other.color;
   this->baseColor = other.baseColor;

   this->OnMouseHover = other.OnMouseHover;
   this->OnMouseOver = other.OnMouseOver;
   this->OnMouseClick = other.OnMouseClick;

   this->title = other.title;

   return *this;
}
```

## ShadersController.cpp

```
#include "ShadersController.h"
#include <glm/gtc/type_ptr.hpp>

std::unique_ptr<
   std::unordered_map<GLuint, std::unique_ptr<Shader>>
> ShadersController::shaders = std::make_unique<std::unordered_map<GLuint, std::unique_ptr<Shader>>>();

void ShadersController::LoadShader(std::unique_ptr<Shader> shader)
{
   shaders->operator[](shader->GetID()) = std::move(shader);
}

void ShadersController::LoadShader(std::string title, std::string vertexPath, std::string fragmentPath)
{
   std::unique_ptr<Shader> shader = std::make_unique<Shader>(title, vertexPath, fragmentPath);
   shaders->operator[](shader->GetID()) = std::move(shader);
}
```

```
Shader* ShadersController::GetShader(GLuint id)
{
   auto it = shaders->find(id);
   if (it != shaders->end())
      return it->second.get();
   return nullptr;
}

const GLuint& ShadersController::GetShaderID(std::string_view title)
{
   for (auto& shader : *shaders)
      if (shader.second->GetTitle() == title)
         return shader.first;
   return -1;
}

void ShadersController::Use(GLuint id)
{
   glUseProgram(id);
}

void ShadersController::Disable()
{
   glUseProgram(0);
}

void ShadersController::SetBool(const GLuint& id, const char* name, bool value)
{
   glUniform1i(glGetUniformLocation(id, name), (int)value);
}

void ShadersController::SetInt(const GLuint& id, const char* name, int value)
{
   glUniform1i(glGetUniformLocation(id, name), value);
}

void ShadersController::SetFloat(const GLuint& id, const char* name, float value)
{
   glUniform1f(glGetUniformLocation(id, name), value);
}

void ShadersController::SetVec2(const GLuint& id, const char* name, float x, float y)
{
   glUniform2f(glGetUniformLocation(id, name), x, y);
}

void ShadersController::SetVec3(const GLuint& id, const char* name, float x, float y, float z)
{
   glUniform3f(glGetUniformLocation(id, name), x, y, z);
```

```
}

void ShadersController::SetVec4(const GLuint& id, const char* name, float x, float y, float z, float
w)
{
   glUniform4f(glGetUniformLocation(id, name), x, y, z, w);
}

void ShadersController::SetMat4(const GLuint& id, const char* name, glm::mat4& matrix)
{
   glUniformMatrix4fv(glGetUniformLocation(id, name), 1, GL_FALSE, glm::value_ptr(matrix));
}

void ShadersController::SetColor(const GLuint& id, const char* name, Color color)
{
   SetVec4(id, name, color.r, color.g, color.b, color.a);
}
```

**RenderInstance.cpp**

```
#include "RenderInstance.h"
#include <glad/glad.h>

void RenderInstance::UpdateVertices()
{
   GenerateVertices();
   if (!VBO || !VAO || !EBO) {
      return Initialize();
   };

   glBindBuffer(GL_ARRAY_BUFFER, VBO);
   void* ptr = glMapBuffer(GL_ARRAY_BUFFER, GL_WRITE_ONLY);
   if (ptr) {
      memcpy(ptr, vertices.data(), vertices.size() * sizeof(float));
      glUnmapBuffer(GL_ARRAY_BUFFER);
   }
}

void RenderInstance::UpdateVertices(std::vector<float> vertices)
{
   this->vertices = vertices;
   if (!VBO || !VAO || !EBO) {
      return Initialize();
   };

   glBindBuffer(GL_ARRAY_BUFFER, VBO);
   void* ptr = glMapBuffer(GL_ARRAY_BUFFER, GL_WRITE_ONLY);
   if (ptr) {
      memcpy(ptr, vertices.data(), vertices.size() * sizeof(float));
      glUnmapBuffer(GL_ARRAY_BUFFER);
   }
```

```
}

RenderInstance::RenderInstance()
   : VBO(0), VAO(0), EBO(0)
{
}

RenderInstance::RenderInstance(unsigned int VBO, unsigned int VAO, unsigned int EBO)
   : VBO(VBO), VAO(VAO), EBO(EBO)
{
}

void RenderInstance::SetVBO(unsigned int& VBO)
{
   this->VBO = VBO;
}

void RenderInstance::SetVAO(unsigned int& VAO)
{
   this->VAO = VAO;
}

void RenderInstance::SetEBO(unsigned int& EBO)
{
   this->EBO = EBO;
}

const unsigned int& RenderInstance::GetVBO()
{
   return VBO;
}

const unsigned int& RenderInstance::GetVAO()
{
   return VAO;
}

const unsigned int& RenderInstance::GetEBO()
{
   return EBO;
}

const std::vector<float>& RenderInstance::GetVertices()
{
   return vertices;
}

void RenderInstance::SetVertices(std::vector<float> vertices)
{
   this->vertices = vertices;
```

```
}

bool RenderInstance::IsInitialized()
{
   return VAO && VBO && EBO;
}

bool RenderInstance::operator==(RenderInstance& other)
{
   return VAO = other.VAO && VBO == other.VBO && EBO == other.EBO &&
      vertices == other.vertices;
}
```

# Додаток В

## Класи живих істот

### **Pawn.cpp**

```
#include "Pawn.h"
#include "../render/RectRenderInstance.h"
#include "../utilities/ptrs.h"
#include "../../Content/Scripts/AI/Movements/MovementsController.h"

void Pawn::MathPos(std::vector<Coord> vertexes)
{
   Coord& vertex1 = vertexes[0];
   Coord& vertex2 = vertexes[1];

   size = Size(vertex2.X - vertex1.X, vertex2.Y - vertex1.Y);

   float centerX_GL = float(vertex1.X + vertex2.X) / 2.0f;
   float centerY_GL = float(vertex1.Y + vertex2.Y) / 2.0f;

   renderInstance->SetVertexes(vertex1, vertex2);

   position = Coord(((centerX_GL + 1.0f) / 2.0f) *
(float)Window::GetRenderResolution().GetWidth(),
      ((1.0f - (centerY_GL + 1.0f) / 2.0f) * (float)Window::GetRenderResolution().GetHeight()));
}

void Pawn::MathPos(Coord& position)
{
   this->position = position;

   float glCenterX = Window::PixelToGLX(position.X);
   float glCenterY = Window::PixelToGLY(position.Y);

   float glWidth = (float)(size.GetWidth()) / (float)Window::GetRenderResolution().GetWidth() *
2.0f;
   float glHeight = (float)size.GetHeight() / (float)Window::GetRenderResolution().GetHeight() *
2.0f;

   float halfWidth = glWidth / 2.0f;
   float halfHeight = glHeight / 2.0f;

   Coord vertex1, vertex2;

   vertex1.X = glCenterX - halfWidth;
   vertex1.Y = glCenterY - halfHeight;

   vertex2.X = glCenterX + halfWidth;
   vertex2.Y = glCenterY + halfHeight;

   renderInstance->SetVertexes(vertex1, vertex2);
```

```
}

void Pawn::MathSize(Size& size)
{
   this->size = size;
   MathPos(position);
}

bool Pawn::MouseInRect(Mouse& mouse)
{
   const Coord& vertex1 = renderInstance->GetVertex1();
   const Coord& vertex2 = renderInstance->GetVertex2();

   float normMouseX = (mouse.GetMouseCoord().X / Window::GetRenderResolution().GetWidth()) * 2.0f - 1.0f;
   float normMouseY = 1.0f - (mouse.GetMouseCoord().Y / Window::GetRenderResolution().GetHeight()) * 2.0f;

   return (normMouseX >= vertex1.X && normMouseX <= vertex2.X &&
      normMouseY >= vertex1.Y && normMouseY <= vertex2.Y);
}

Pawn::Pawn(
   std::string title,
   CharacterTypes characterType,
   std::shared_ptr<ICollision> collision, std::shared_ptr<Material> material, Directions moveDirection,
   Coord position, Size size, float speed, float maxSpeed, float minSpeed,
   float health, float maxHealth, bool isPlayable, bool isKinematic, bool isHidden,
   std::vector<std::shared_ptr<IAnimation>> animations
){

   SetTitle(title);

   this->characterType = characterType;

   renderInstance = std::make_unique<RectRenderInstance>(std::move(material));

   this->collision = collision;

   this->size = size;
   MathPos(position);

   this->moveDirection = moveDirection;

   this->speed = speed;
   this->maxSpeed = maxSpeed;
   this->minSpeed = minSpeed;

   this->health = health;
```

```
    this->maxHealth = maxHealth;

    this->isPlayable = isPlayable;
    this->kinematic = isKinematic;
    this->isHidden = isHidden;

    target = {};

    SetLayer(Layer::Pawn);

    SetAction(Actions::Idle);

    this->animations.AddAnimations(animations);
}

void Pawn::SetMoveDirection(Directions moveDirection)
{
    this->moveDirection = moveDirection;
}

void Pawn::SetPos(std::vector<Coord> vertexes, bool render)
{
    MathPos(vertexes);

    if (!render) {
        return;
    }

    UpdateVertices();
}

void Pawn::SetPos(Coord position, bool render)
{
    MathPos(position);

    if (!render) {
        return;
    }

    UpdateVertices();
}

void Pawn::RotateToDirection(Directions direction)
{
}

void Pawn::SetTitle(std::string title)
{
    this->title = title;
```

```
}

void Pawn::SetSize(Size size, bool render)
{
   MathSize(size);

   if (!render) {
      return;
   }

   UpdateVertices();
}

void Pawn::SetSpeed(float speed)
{
   this->speed = speed;
}

void Pawn::SetMaxSpeed(float maxSpeed)
{
   this->maxSpeed = maxSpeed;
}

void Pawn::SetMinSpeed(float minSpeed)
{
   this->minSpeed = minSpeed;
}

void Pawn::SetHealth(float health)
{
   this->health = health;
}

void Pawn::SetMaxHealth(float maxHealth)
{
   this->maxHealth = maxHealth;
}

void Pawn::SetWeight(float weight)
{
   this->weight = weight;
}

void Pawn::SetIsPlayable(bool isPlayable)
{
   this->isPlayable = isPlayable;
}

bool Pawn::IsKinematic()
```

```
{
    return kinematic;
}

void Pawn::SetKinematic(bool kinamatic)
{
    this->kinematic = kinamatic;
}

void Pawn::SetIsHidden(bool isHidden)
{
    this->isHidden = isHidden;
}

void Pawn::SetCollision(std::shared_ptr<ICollision> collision)
{
    this->collision = collision;
}

void Pawn::SetMaterial(std::shared_ptr<Material> material)
{
    renderInstance->SetMaterial(material);
}

void Pawn::SetColor(Color color)
{
    if (!ValidWeakPtr<Material>(renderInstance->GetMaterial())) {
        return;
    }

    renderInstance->GetMaterial().lock()->SetDiffuse(color);
}

void Pawn::Damage(float damage)
{
    if (health <= 0) {
        return;
    }
    health -= damage;
}

void Pawn::Die()
{
    health = 0;
}

bool Pawn::MouseHover(Mouse& mouse)
{
    if (mouse.IsEqual()) {
```

```
        return false;
    }

    const bool isHover = MouseInRect(mouse);

    if (isHover && OnMouseHover) {
        OnMouseHover(this, Window::GetWindow());
    }

    if (!isHover && OnMouseOver) {
        OnMouseOver(this, Window::GetWindow());
    }

    return isHover;
}

bool Pawn::MouseClick(Mouse& mouse)
{
    if (!mouse.isClick() || !MouseInRect(mouse)) {
        return false;
    }

    OnMouseClick(this, Window::GetWindow());
    return true;
}

bool Pawn::CollisionEnter(IGameObject& gameObject)
{
    if (!collision || collision == nullptr) {
        return false;
    }

    bool isEnter = collision->IsCollisionEnter(&gameObject);

    if (isEnter && OnCollisionEnterHandler) {
        OnCollisionEnterHandler(this, &gameObject, Window::GetWindow());
    }
}

void Pawn::SetTarget(std::weak_ptr<Pawn> target)
{
    this->target = std::move(target);
}

std::weak_ptr<class Pawn> Pawn::GetTarget()
{
    return target;
}
```

```
void Pawn::HookMouseHover(MouseHoverHandler onMouseHover)
{
   this->OnMouseHover = onMouseHover;
}

void Pawn::HookMouseOver(MouseHoverHandler onMouseOver)
{
   this->OnMouseOver = onMouseOver;
}

void Pawn::HookMouseClick(MouseClickHandler onMouseClick)
{
   this->OnMouseClick = onMouseClick;
}

void Pawn::HookOnCollisionEnter(OnCollisionEnter onCollisionEnter)
{
   OnCollisionEnterHandler = onCollisionEnter;
}

const Coord& Pawn::GetPos()
{
   return position;
}

Coord Pawn::GetStartPos()
{
   return startPos;
}

const Coord& Pawn::GetOpenGlPos()
{
   return Coord(Window::PixelToGLX(position.X), Window::PixelToGLY(position.Y));
}

Size Pawn::GetSize()
{
   return size;
}

Color Pawn::GetColor()
{
   return ValidWeakPtr<Material>(renderInstance->GetMaterial())
      ? renderInstance->GetMaterial().lock()->GetDiffuse()
      : Color();
}

Offset& Pawn::GetOffset()
{
```

```
    return offset;
}

std::weak_ptr<Material> Pawn::GetMaterial()
{
    return renderInstance->GetMaterial();
}

Directions Pawn::GetMoveDirection()
{
    return moveDirection;
}

std::weak_ptr<ICollision> Pawn::GetCollision()
{
    return collision;
}

Color Pawn::GetBaseColor()
{
    return ValidWeakPtr<Material>(renderInstance->GetMaterial())
        ? renderInstance->GetMaterial().lock()->GetDiffuse()
        : Color();
}

AnimationController& Pawn::GetAnimations()
{
    return animations;
}

Layer Pawn::GetLayer()
{
    return layer;
}

void Pawn::SetLayer(Layer layer)
{
    this->layer = layer;
}

Actions Pawn::GetAction()
{
    return action;
}

void Pawn::SetAction(Actions action)
{
    this->action = action;
}
```

```
float Pawn::GetSpeed()
{
   return speed;
}

float Pawn::GetMaxSpeed()
{
   return maxSpeed;
}

float Pawn::GetMinSpeed()
{
   return minSpeed;
}

float Pawn::GetViewDistance()
{
   return viewDistance;
}

float Pawn::GetViewWidth()
{
   return viewWidth;
}

std::string_view Pawn::GetTitle()
{
   return title;
}

const std::string& Pawn::GetTitleString()
{
   return title;
}

float Pawn::GetHealth()
{
   return health;
}

float Pawn::GetMaxHealth()
{
   return maxHealth;
}

float Pawn::GetWeight()
{
   return weight;
```

```
}

float Pawn::GetDamageDistance()
{
   return damageDistance;
}

bool Pawn::GetIsPlayable()
{
   return isPlayable;
}

bool Pawn::GetIsHidden()
{
   return isHidden;
}

const bool Pawn::IsNear(Coord startPos, Coord targetPos, float distance)
{
   return (std::abs(startPos.X) == std::abs(targetPos.X)
      && std::abs(startPos.Y) == std::abs(targetPos.Y)) ||
      (std::abs(startPos.X - targetPos.X) <= distance &&
         std::abs(startPos.Y - targetPos.Y) <= distance);
}

const bool Pawn::IsMouseOverlap()
{
   return MouseInRect(*Window::GetMouse().lock());
}

const bool Pawn::IsDead()
{
   return health <= 0;
}

const bool Pawn::IsWalkable(Coord& position)
{
   return CheckForCollision(position);
}
```