

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка 3D відеогри "Whispers of the Wildwood" в
жанрі виживання з елементами RPG. Спец-частина: розробка
Frontend-частини системи»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

*Кваліфікаційна робота містить результати власних досліджень. Використання
ідей, результатів і текстів інших авторів мають посилання
на відповідне джерело*

(підпис) Вероніка РОМАШКО

Виконала: здобувачка вищої освіти групи ПД-43

Вероніка РОМАШКО

Керівник: _____
д-р техн. наук, проф. Ірина ЗАМРІЙ

Рецензент: _____

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**
Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____ Ірина ЗАМРІЙ

« ____ » _____ 2026 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Ромашко Вероніці Вадимівні

1. Тема кваліфікаційної роботи: «Розробка 3D відеогри "Whispers of the Wildwood" в жанрі виживання з елементами RPG. Спец-частина: розробка Frontend-частини системи»

керівник кваліфікаційної роботи д-р техн. наук, професор Ірина ЗАМРІЙ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи розробки відеоігор, опис методів розробки відеоігор на рушіях, технічна документація з описом бібліотек для обробки розробки відеоігор.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз frontend-частини в існуючих відеоігор в жанрі виживання з елементами RPG та рушіїв для розробки frontend-частини.

2. Проектування frontend-частини відеоігори в жанрі виживання з елементами RPG.

3. Програмна реалізація та опис функціонування frontend-частини відеоігри в жанрі виживання з елементами RPG

4. Тестування frontend-частини відеогри.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.

2. Вимоги до програмного забезпечення.

3. Програмні засоби реалізації.

4. Діаграма варіантів використання.

5. Алгоритм роботи відеогри.

6. Діаграма класів.

7. Екранні форми.

8. Апробація результатів дослідження

6. Дата видачі завдання «23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	23.02-28.02.2026	
2	Аналіз та дослідження існуючих аналогів	01.03-07.03.2026	
3	Огляд frontend-частини в існуючих відеоігор та методів їх розробки	08.03-18.03.2026	
4	Проектування frontend-частини відеоігри	19.03-28.03.2026	
5	Програмна реалізація frontend-частини відеоігри	28.03-20.04.2026	
6	Тестування frontend-частини відеогри	20.04-25.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	26.04-01.05.2026	
8	Розробка демонстраційних матеріалів	02.05-08.05.2026	
9	Попередній захист роботи	11.05-22.05.2026	

Здобувач(ка) вищої освіти

(підпис)

Вероніка РОМАШКО

Керівник

кваліфікаційної роботи

(підпис)

Ірина ЗАМРІЙ

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 64 стор., 4 табл., 22 рис., 21 джерел.

Мета роботи – покращення ігрового досвіду користувача за рахунок поєднання геймплею жанру виживання з ключовими елементами RPG.

Об'єкт дослідження – процес набуття досвіду під час дослідження механік відеогри у жанрі виживання з елементами RPG.

Предмет дослідження – програмні засоби та методи створення геймплею відеогри у жанрі виживання з елементами RPG.

Короткий зміст роботи: Проведено аналіз жанру виживання з RPG елементами, визначена його цільова аудиторія та особливості frontend-частини в цьому жанрі в поєднанні з RPG елементами. Проаналізовано ігри-аналоги жанру виживання з елементами RPG: Minecraft, The Forest, Rust, ARK: Survival Evolved та Don't Starve. Розроблено алгоритм роботи відеогри та програмно реалізовані ключові функціональні можливості frontend-частини, зокрема: система інвентарю, головного меню, відображення станів персонажа, а також була розроблена мапа ігрового світу з кількома локаціями. Проведено функціональне та модульне тестування frontend-частини відеогри. В роботі використано мову програмування C#, ігровий рушій Unity та середовище розробки Visual Studio.

Сферою використання відеогри є розвага, отримання досвіду, відпочинок та проведення гарного часу за проходженням.

КЛЮЧОВІ СЛОВА: FRONTEND, ВИЖИВАННЯ, RPG, UNITY, ІНТЕРФЕЙС, ДИЗАЙН, РОЗРОБКА, C#.

ЗМІСТ

ВСТУП.....	10
1 ДОСЛІДЖЕННЯ FRONTEND-СКЛАДОВОЇ У ІГРАХ ЖАНРУ ВИЖИВАННЯ З RPG-ЕЛЕМЕНТАМИ.....	12
1.1 Поняття жанру виживання з елементами RPG та його ключові характеристики	12
1.2 Роль і значення frontend-складової у відеоіграх	14
1.3 Характеристика цільової аудиторії.	15
1.4 Аналіз frontend-реалізації у популярних іграх-аналогах	16
1.4.1 Minecraft.....	16
1.4.2 The Forest	17
1.4.3 Rust	19
1.4.4 ARK: Survival Evolved	20
1.4.5 Don't Starve.....	22
1.4.6 Порівняльний аналіз frontend-реалізації у популярних іграх-аналогах ...	24
1.5 Форумвання функціональних та нефункціональних вимог до відеогри.....	26
1.5.1 Функціональні вимоги.....	26
1.5.2 Нефункціональні вимоги.....	27
2 ЗАСОБИ РЕАЛІЗАЦІЇ FRONTEND-ЧАСТИНИ ВІДЕОГРИ	28
2.1 Мова програмування C#.....	28
2.1.1 Переваги та недоліки C#	28
2.2 Огляд рушія Unity у контексті розробки інтерфейсу	30
2.2.1 Характеристика можливостей рушія Unity у реалізації frontend-частини	31
2.2.2 Переваги та недоліки рушія Unity у frontend-розробці.....	31
2.2.3 Особливості використання мови програмування C# при розробці frontend-частини в Unity	32
2.3 Використання середовища розробки Visual Studio у поєднанні з рушієм Unity	33
2.3.1 Характеристика середовища розробки Visual Studio	33
2.3.2 Інтеграція Visual Studio з рушієм Unity	34
3 ПРОЄКТУВАННЯ ТА РОЗРОБКА FRONTEND-ЧАСТИНИ ВІДЕОГРИ	36

3.1 Розробка візуального дизайну та стилю гри	36
3.2 Проектування ігрового світу (мапи)	37
3.2.1 Містечко з NPC	39
3.2.2 Ферма	40
3.2.3 Цвинтар	41
3.2.4 Закинутий табір	42
3.3 Розробка меню	43
3.3.1 Головне меню	43
3.3.2 Меню паузи.....	46
3.4 Реалізація системи інвентарю гравця	48
3.5 Розробка відображення станів персонажа.....	55
4 ТЕСТУВАННЯ FRONTEND-ЧАСТИНИ ВІДЕОГРИ	58
4.1 Тестування ігрового світу (мапи)	58
4.2 Тестування користувацького інтерфейсу	60
ВИСНОВКИ.....	63
ПЕРЕЛІК ПОСИЛАНЬ	65
ДОДАТОК А ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ	68
ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ.....	81

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

UI – User Interface

UX – User Experience

RPG – Role-Playing Game

HUD – Head-Up Display

PvP – Player vs Player

GUI – Graphical User Interface

VS – Visual Studio

ПЗ – Програмне забезпечення

NPC – Non-Player Character

RAM – Random Access Memory

FPS – Frames Per Second

ВСТУП

Актуальність теми – на сьогоднішній день, індустрія відеоігор є однією з найбільш розвинутих та прибуткових галузей цифрових технологій, що активно набуває популярність серед молоді та навіть старшого покоління. Велику прихильність серед користувачів здобули 3D-відеоігри у жанрі survival-RPG, серед механік цього жанру користувачі виділяють: відкритий ігровий світ, дослідження територій, систему розвитку персонажа та взаємодію з навколишнім середовищем. У зв'язку з цим зростає потреба у розробленні frontend-частини для таких проєктів, саме щоб здатних забезпечити стабільну роботу, високу продуктивність та зручний користувацький інтерфейс.

Доцільність проведення дослідження пояснюється потребою у створенні frontend-частини 3D-відеоігри, яка відповідає за взаємодію користувача з ігровим середовищем, відображення графічного інтерфейсу (GUI), обробку візуальних елементів та реалізацію основних механік керування. Зростання вимог до візуалу та зручності ігрових інтерфейсів робить зв'язку Unity, C# та Visual Studio незамінним стандартом сучасної розробки. Водночас, дедалі складніша ігрова система і кількості інтерактивних об'єктів потребує оптимізації візуальної частини, ефективного управління ресурсами та забезпечення комфортної взаємодії гравця з ігровим світом. Ось чому розробка frontend-частини є актуальним напрямом дослідження в індустрії створення відеоігор.

Мета роботи – покращення ігрового досвіду користувача за рахунок поєднання геймплею жанру виживання з ключовими елементами RPG.

Об'єкт дослідження – процес набуття досвіду під час дослідження механік відеоігри у жанрі виживання з елементами RPG.

Предмет дослідження – програмні засоби та методи створення геймплею відеоігри у жанрі виживання з елементами RPG.

Методи дослідження – для досягнення поставленої мети у роботі було використано комплекс методів дослідження, зокрема проведено аналіз предметної

галузі та ринку відеоігор у survival-RPG, здійснено огляд і порівняльний аналіз ігор-аналогів, досліджено важливість frontend-частини ігрових систем, а також методи реалізації користувацького інтерфейсу та взаємодії гравця з ігровим середовищем. На основі проведеного аналізу було обрано засоби реалізації відеогри, зокрема мову програмування C#, середовище розробки Visual Studio та рушій Unity.

Наукова новизна роботи – визначається поєднанням механік жанру виживання та RPG у межах відеогри, що включає реалізацію сюжетної квестової системи, інтерактивну взаємодію гравця з ігровим середовищем та дослідження відкритого світу. Особливістю проєкту є інтеграція frontend-компонентів із динамічними ігровими механіками, що забезпечують відображення квестів, діалогів, інвентарю та елементів UI/UX у реальному часі.

Практична значущість результатів – полягає в можливості користувача отримати приємний досвід та розвагу під час проходження гри.

1 ДОСЛІДЖЕННЯ FRONTEND-СКЛАДОВОЇ У ІГРАХ ЖАНРУ ВИЖИВАННЯ З RPG-ЕЛЕМЕНТАМИ

1.1 Поняття жанру виживання з елементами RPG та його ключові характеристики

Сьогодні жанр виживання (survival) активно набирає популярність серед гравців у всьому світі, особливо у поєднанні з RPG-елементами. І це досить зрозумілий вибір, бо такі відеоігри поєднують напружений процес виживання в умовах браку ресурсів із механіками розвитку персонажа. Геймплей зосереджений на виживанні у небезпечному середовищі, де гравець мусить стежити за станом героя: від рівня здоров'я до голоду й температури тіла. Елементи RPG гармонійно доповнюють цей процес, додаючи систему прогресії з рівнями, вміннями та можливістю налаштування персонажа під свій стиль. Дослідники стверджують: рольові ігри вирізняються серед інших завдяки системі розвитку персонажа та можливості приймати рішення. [1].

Комбінація цих двох жанрів формує цілісну та складну систему, що дозволяє гравцеві поринути в ігровий світ. Користувач повинен не тільки швидко реагувати на небезпеку, а й планувати свої дії так щоб зменшити поживання ресурсів та не дати показникам свого персонажа падати занадто низько. Ключовими елементами гри є вивчення відкритого простору та менеджмент ресурсів. Особливої глибини процесу додають динамічна зміна погоди й добовий цикл, які безпосередньо коригують ігрові механіки.

Другою з ключових механік є система крафтингу, що дозволяє створювати предмети із зібраних ресурсів. Вона може складатися як і з простих рецептів, де використовуються звичайні предмети які легко добути, так і з складних рецептів в яких більш складні умови. Разом із цим важливе значення має управління інвентарем, оскільки обмеженість ресурсів змушує гравця приймати стратегічні рішення щодо їх використання. Досвідчені геймдизайнери вважають обмеження

ресурсів ключовим механізмом, який змушує гравців думати креативно та прораховувати свої дії наперед [2].

Елементи рольових ігор поглиблюють досвід виживання, роблячи ставку на прогресію та розвиток персонажа. Система досвіду, рівнів і навичок дозволяє покращувати характеристики героя. Контроль над кастомізацією та прогресом персонажа безпосередньо формує мотивацію гравця, що є визначальною рисою рольових ігор на думку експертів [3]. Також важливою складовою цього жанру є дерево навичок чи інші системи розвитку, які дають змогу обирати стиль проходження гри, що підвищує реіграбельність, оскільки різні комбінації навичок можуть суттєво змінювати ігровий процес.

Як виживання, так і RPG мають багато переваг, які допомагають їм доповнювати один одного. Складні умови виживання підштовхують гравця розвиватися та відкривати нові можливості для ефективнішого проходження гри. Це створює відчуття поступового прогресу та розвитку персонажа. Теоретична база ігор стверджує: ми отримуємо задоволення не від самої перемоги, а від навчання. Мозок винагороджує нас за кожен крок у вдосконаленні навичок та подоланні перешкод [4].

Система бою та виживання в агресивному середовищі є ще одним критичним аспектом. Небезпеки поділяються на дві категорії: прямі (супротивники та агресивні істоти) та опосередковані (дефіцит ресурсів чи суворий клімат). RPG-складова дозволяє боротися з цими проблемами, розвиваючи навички, використовуючи спорядження і варіанти тактики або прокачування. У багатокористувацьких режимах додаткову роль відіграє взаємодія між гравцями, яка може бути кооперативною або конкурентною.

Як підсумок, поєднання цих двох жанрів, дозволяє гравцеві відчути механіки виживання з комбінацією розвитку персонажа, що створює цікавий та захопливий геймплей зі змогою реіграбельності. Виживання та прокачування гармонійно поєднуються, створюючи гармонійний ігровий процес з цікавим досвідом.

1.2 Роль і значення frontend-складової у відеоіграх

Роль фронтенду в ігровій індустрії важко переоцінити, адже він є сполучною ланкою між людиною та кодом. Саме вона відповідає за відображення інформації, керування та загальний UX. На відміну від backend-частини, яка обробляє внутрішню логіку, frontend виступає безпосереднім інтерфейсом, через який людина бачить відеогру і взаємодіє з нею. Від того, наскільки якісно це зроблено, залежить зручність і легкість занурення у гру.

Однією з пріоритетних функцій клієнтської частини (frontend) є рендеринг графічного інтерфейсу. Сюди входять усі інтерактивні компоненти, які забезпечують доступ до даних і взаємодію з ігровим середовищем:

- меню гри та навігаційні елементи;
- панелі стану та показників персонажа;
- інвентар і система взаємодії з предметами;
- ігрову мапу, підказки та журнали завдань.

Проектування цих компонентів має забезпечувати миттєву орієнтацію та легкий доступ до даних. Експерти з геймдизайну наголошують, що саме від продуманості інтерфейсу та способу візуалізації інформації залежить, наскільки глибоко гравець «пірне» в ігровий світ [5].

Окрім виведення даних, frontend бере на себе опрацювання дій гравця. Він інтегрує взаємодію через клавіатуру, мишу або контролер, де критично важливими є швидкість реакції та коректність виконання кожної команди. Це особливо важливо в динамічних іграх, де навіть невеликі затримки можуть зіпсувати ігровий процес. Згідно з теорією геймдизайну, фундаментальними засадами проектування інтерфейсів є забезпечення гравця відчуттям контролю та надання миттєвого зворотного зв'язку [2].

Ще одним аспектом є візуальна частина фронтеду. Сюди належать візуальна стилістика, графічні ефекти, анімації та дизайн інтерфейсу. Науковці підкреслюють, що візуальна складова виходить за межі чистої естетики: вона

служує стратегічним каналом комунікації, що визначає ергономічність ігрового процесу та динаміку когнітивних реакцій гравця [6]. Адже саме через візуалізацію користувач сприймає ігровий світ, тому навіть відповідність елементів інтерфейсу щодо певного стилю до вибраної тематики є дуже важливим. Емоційний вплив візуальної естетики також здатний підсилювати мотивацію гравця та його позитивний досвід взаємодії з грою [7].

Фронтенд-складова є фундаментальною, бо поєднує візуальну презентацію, функціональність та інтерфейс. Її належна реалізація не лише підвищує комфорт використання, а й безпосередньо впливає на лояльність користувача та якість ігрового досвіду.

1.3 Характеристика цільової аудиторії.

Цільова аудиторія ігор цього жанру досить різноманітна, але її можна розділити за основними ознаками: вік, ігрові вподобання, досвід і те, чого люди очікують від гри. Більшість гравців – це люди від 10 до 35 років. Вони добре знайомі з сучасними технологіями, мають досвід у відеоіграх і люблять різноманітні механіки. Водночас останнім часом розробники все активніше залучають і старшу аудиторію, адже жанр приваблює цікавим геймплеєм та можливістю приймати власні важливі рішення.

З ігрової точки зору ці люди віддають перевагу відкритим світам, нелінійним сюжетам, прокачці персонажа і виживанню. Вони хочуть бачити інтуїтивний, але при цьому функціональний інтерфейс, який дозволяє швидко розуміти стан героя, інвентар, мапу та що можна зробити далі. Казуальні гравці віддають перевагу простому інтерфейсу, швидкому входженню в гру і зрозумілій навігації. Хардкорні ж гравці, навпаки, люблять складні й детальні системи, навіть якщо через це інтерфейс стає більш навантаженим. Для створення рівноцінно зручного досвіду для всіх категорій користувачів, архітектура інтерфейсу має бути максимально адаптивною та гнучкою.

Оскільки запити аудиторії у жанрі survival-RPG є досить різними, розробка фронтенду має базуватися на глибоко продуманому функціоналі, інтуїтивно зрозумілому UX та естетично довершеному візуалі.

1.4 Аналіз frontend-реалізації у популярних іграх-аналогах

Аналіз актуальних представників жанру дозволяє виокремити ключові стратегії розробки UI, зважити їхні переваги та недоліки, а також проаналізувати дієвість мінімалістичних, дієгетичних чи комплексних рішень. Це також дозволяє виявити слабкі місця в юзабіліті, як-от перевантаження даними чи неінтуїтивна навігація, що часто стають проблемою в іграх зі складними ігровими системами.

Для аналізу варто врахувати контекст використання інтерфейсу: динамічність ігрового процесу, мультиплеєрну взаємодію, складність механік і очікування цільової аудиторії. Це дуже сильно впливає на вимоги до frontend-реалізації.

1.4.1 Minecraft

В основі інтерфейсу Minecraft лежать читабельність та впізнаваний стиль. UI наслідує загальну «блокову» естетику гри, використовуючи піксельні текстури та просту геометрію. Це створює цілісний користувацький досвід, де елементи керування органічно вписуються в оточення, не порушуючи занурення.

Завдяки використанню спрощених іконок користувач сприймає інформацію без затримок. Логічне структурування інтерфейсу з акцентом на нижній частині екрана мінімізує візуальний шум і не перекриває ігрову сцену [8].

Переваги:

- мінімалістичний дизайн, що не перевантажує екран;
- висока читабельність завдяки простим іконкам і контрастності;
- цілісність стилю інтерфейсу та ігрового світу;
- зручне розташування елементів, що не заважає геймплею;

- легкість освоєння для більшості гравців.

Недоліки:

- відсутність детальних підказок може ускладнювати початок гри для новачків;
- система крафтингу потребує запам'ятовування рецептів або використання зовнішніх джерел;
- обмежена інформативність у порівнянні з більш складними відеоіграми жанру виживання;
- мінімалізм інтерфейсу іноді не відображає повний стан ігрових систем.



Рис. 1.1 Інтерфейс користувача в Minecraft

1.4.2 The Forest

The Forest є представником дієгетичного інтерфейсу. Тобто, це коли UI максимально інтегрований в ігровий світ і не вибивається з сприйняття гри. Візуальний стиль відповідає атмосфері: реалістичне оточення, приглушені кольори та відсутність яскравих елементів, що підсилює ефект пригніченості.

оловна «фішка» The Forest – це максимальна реалістичність взаємодії, реалізована через дієгетичний інвентар-рюкзак. амість абстрактних меню, гравець

маніпулює предметами безпосередньо в ігровому просторі, що посилює іммерсивність. Інформація про стан персонажа подається через невеликі індикатори або поведінкові сигнали [9].

Завдяки ненав'язливому розміщенню UI-елементів, екран залишається незасміченим. Контекстна поява підказок дозволяє гравцеві повністю сконцентруватися на дослідженні оточення, а не на читанні меню.

Переваги:

- інтерфейс, що не перевантажує екран і сприяє зануренню;
- високий рівень інтеграції UI в ігровий світ;
- реалістична взаємодія з інвентарем і предметами;
- контекстні підказки, що з'являються лише за необхідності;
- сильний вплив на атмосферу та емоційне сприйняття гри.

Недоліки:

- обмежена інформативність через відсутність детальних показників;
- складність освоєння для новачків через мінімальну кількість підказок;
- повільніша взаємодія з інвентарем у порівнянні з класичними меню;
- відсутність чіткої структури може ускладнювати навігацію;
- менша ефективність у динамічних або бойових ситуаціях.



Рис. 1.2 Рюкзак (інвентар) та крафт в The Forest

1.4.3 Rust

В основі UI-дизайну Rust лежить баланс між прагматизмом та інформаційним наповненням. Естетика гри розкривається через стриману гаму та «невагомі» напівпрозорі панелі. Кожен елемент тут працює на створення єдиного стилю, що органічно вписується у жорсткий та реалістичний світ, де кожна секунда зволікання може коштувати життя.

Його UI постійно відображає важливі параметри персонажа, такі як здоров'я, голод і спрага. Графічні індикатори спрощують моніторинг ігрового стану, але їхня ефективність напряду залежить від навичок і наслуханості гравця.

Інтерфейс користувача (HUD) у Rust зосереджений у нижній частині екрана, тоді як менеджмент ресурсів та крафтинг винесені в окремі вікна. Ключовою особливістю є інвентар із сітковою системою, він дозволяє оперативно маніпулювати предметами, що дає критичну перевагу в динамічних PvP-суперечках. [10].

Переваги:

- функціональний і насичений інтерфейс, що забезпечує доступ до великої кількості інформації;
- ефективна система інвентарю з можливістю швидкої взаємодії з предметами;
- зручна інтеграція крафтингу та управління ресурсами;
- орієнтація на швидкість прийняття рішень у динамічному середовищі.

Недоліки:

- перевантаженість інтерфейсу великою кількістю елементів;
- складність освоєння для новачків через відсутність чіткої візуальної ієрархії;
- інтерфейс крафтингу може бути неінтуїтивним і громіздким;
- велика кількість вікон і вкладок ускладнює навігацію;
- менш виражена стилістична цілісність порівняно з більш художньо орієнтованими іграми.

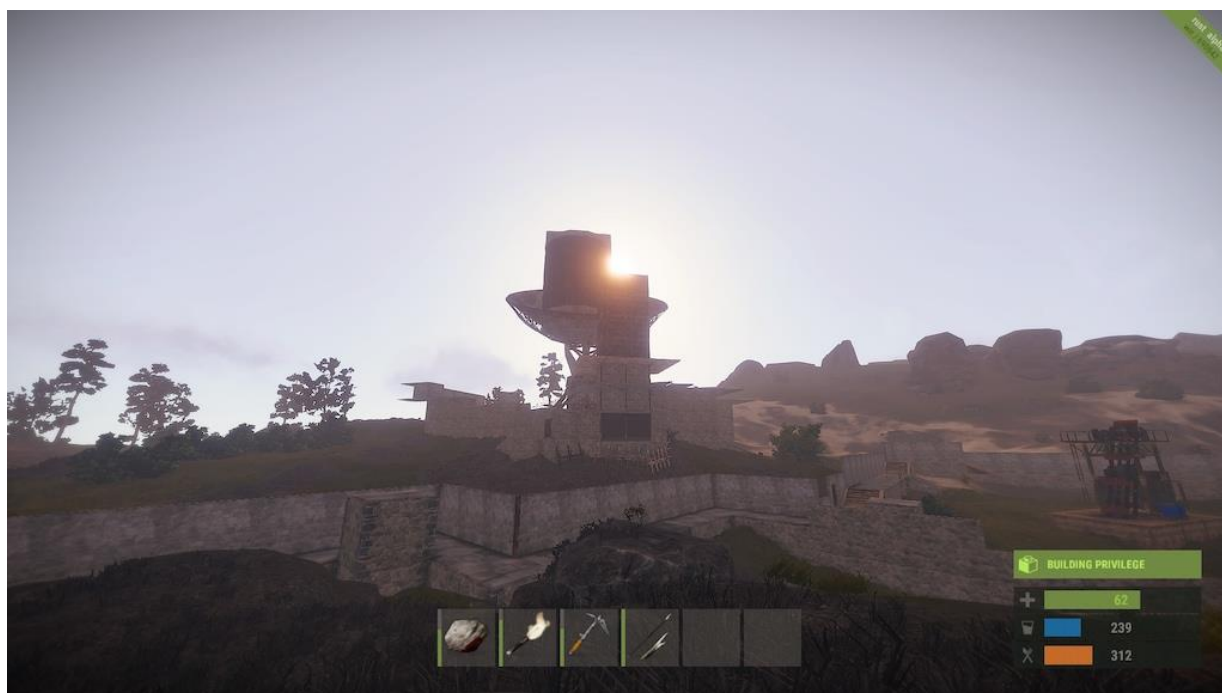


Рис. 1.3 Інтерфейс Rust

1.4.4 ARK: Survival Evolved

ARK: Survival Evolved не намагається бути простою: її інтерфейс максимально інформативний та складний, що відображає глибину внутрішніх систем виживання. Напівреалістичний стиль із напівпрозорими вікнами та великою кількістю тексту може збити новачка з пантелику. Хоча такий підхід забезпечує гравця всіма необхідними даними, він ускладнює освоєння гри на початкових етапах і виглядає як зовнішня надбудова, а не органічна частина ігрового світу.

Геймплей проєкту перенасичений різноманітними метриками: від стану здоров'я та витривалості до рівня гідратації, кисню та температури. Така деталізація приваблює хардкорну аудиторію, проте перевантажений числами й шкалами інтерфейс, хоч і є максимально інформативним, стає серйозним викликом для казуалів.

Інвентар виступає центральним елементом інтерфейсу та реалізований у форматі вкладок, де інтегровано управління ресурсами, крафтинг і прогресію персонажа. Проте надмірна інформативність цього меню створює ризик погіршення користувацького досвіду (UX), зміщуючи фокус із основного геймплею [11].

Переваги:

- висока інформативність інтерфейсу та відображення великої кількості параметрів;
- широкий функціонал інвентаря, що поєднує крафтинг, управління ресурсами та розвиток персонажа;
- гнучкість і масштабованість UI, що дозволяє підтримувати складні ігрові механіки;
- точність відображення стану персонажа завдяки числовим показникам;
- інтеграція різних систем гри в єдиному інтерфейсі.

Недоліки:

- перевантаженість інтерфейсу та високе когнітивне навантаження на гравця;
- складність освоєння для новачків через велику кількість вкладок і параметрів;
- менша візуальна цілісність із ігровим світом порівняно з більш мінімалістичними рішеннями;
- значна частка екрану може займатися меню, що знижує рівень занурення;
- не завжди інтуїтивна навігація між елементами інтерфейсу.



Рис. 1.4 Інтерфейс гравця в ARK: Survival Evolved

1.4.5 Don't Starve

Don't Starve в свою чергу має унікальну стилізацію, що поєднується з функціональністю та не перешкоджає сприйняттю інформації. Елементи гри виконані у готичній мальованій стилістиці з темними відтінками, нерівними контурами та виразними іконками. Це дозволяє інтерфейсу органічно вписуватися в загальний сетинг, додаючи йому особливого шарму, що вигідно вирізняє проєкт серед конкурентів.

Гра зберігає інтуїтивність навіть за високої інформаційної насиченості завдяки продуманому дизайну. Окрім класичних показників здоров'я та ситості, у Don't Starve впроваджено шкалу розсудливості (психічного стану).

Усі ці параметри візуалізовані у вигляді великих стилізованих іконок, що добре зчитуються в динаміці. Компонування екрана також сприяє комфорту: головні дані знаходяться зверху, а система створення речей завжди під рукою ліворуч, що дозволяє крафтити предмети на льоту.

На відмінно від інших, тут меню крафтингу є вертикальне, у якому рецепти згруповані за категоріями. Дане рішення оптимізує навігацію між об'єктами та реалізує механіку поетапного розширення ігрового функціоналу [12].

Переваги:

- стилізований дизайн, що гармонійно поєднується з атмосферою гри;
- висока читабельність завдяки великим іконкам та чіткій візуальній ієрархії;
- зручне вертикальне меню крафтингу з групуванням рецептів;
- швидкий доступ до основних функцій без перевантаження екрана;
- поступове відкриття механік, що зменшує когнітивне навантаження на гравця.

Недоліки:

- велика кількість стилізованих елементів може бути незвичною для нових гравців;
- інтерфейс менш ефективний для відображення складних RPG-систем і великої кількості параметрів;
- деякі елементи можуть виглядати перевантаженими через художню деталізацію;
- обмежена масштабованість інтерфейсу для більш комплексних механік виживання.



Рис. 1.5 Меню крафту в Don't Starve

1.4.6 Порівняльний аналіз frontend-реалізації у популярних іграх-аналогах

Огляд ігор-аналогів свідчить про те, що структура інтерфейсу в кожному проєкті диктується специфікою ігрових механік. Так, у Minecraft UI максимально спрощений для легкості освоєння; у The Forest реалізовано дієгетичний підхід для посилення іммерсивності; Rust фокусується на ергономії даних для інтенсивних PvP-сутичок. Своєю чергою, ARK пропонує багатокomпонентний інтерфейс для управління складними системами, а Don't Starve вдало балансує між унікальною візуальною стилістикою та чіткістю подачі інформації.

Незважаючи на розбіжності у графічній стилістиці та структурі UI, пріоритетом для всіх аналізованих ігор залишається якісна комунікація гравця з ігровим середовищем. Детальний порівняльний аналіз реалізації фронтенду представлено нижче в таблиці 1.1.

Таблиця 1.1

Порівняльний аналіз frontend-реалізації у популярних іграх-аналогах.

Критерій / Гра	Minecraft	The Forest	Rust	ARK: Survival Evolved	Don't Starve
Мінімалістичний інтерфейс	+	+	-	-	+
Висока інформативність	-	-	+	+	+
Дієгетичний UI	-	+	-	-	-
Контекстні підказки	+	+	+	+	+
Простота освоєння	+	-	-	-	+
Не велика кількість параметрів	+	-	-	-	+
Простий UI крафту	+	-	+	+	-
Швидкий доступ до предметів	+	-	+	+	+
Художня стилізація інтерфейсу	+	+	-	-	+
Високий рівень занурення	+	+	-	-	+
Складна структура меню	-	-	+	+	-
Орієнтація на PvP-взаємодію	-	-	+	+	-
Високе когнітивне навантаження	-	-	+	+	-
Масштабованість інтерфейсу	+	-	+	+	-

1.5 Форумвання функціональних та нефункціональних вимог до відеогри

Розробка відеогри починається з визначення вимог, які поділяються на два основних, які поділяються на два види. Функціональні вимоги регламентують безпосередню поведінку системи (її дії): від геймплейних механік і логіки прогресії до роботи інтерфейсу та систем збереження даних. Натомість нефункціональні вимоги встановлюють якісні критерії та експлуатаційні обмеження.

У контексті ігрової індустрії це передусім оптимізація продуктивності, швидкість завантаження, стабільність роботи та адаптивність під різне апаратне забезпечення [13].

У межах проєктування було сформульовано перелік основних вимог до гри, які поділяються на функціональні та нефункціональні, що допоможуть в розробці.

1.5.1 Функціональні вимоги

Згідно з концепцією проєкту, до функціональних вимог, що забезпечують коректну взаємодію гравця з ігровим середовищем, належать такі пункти:

- відображення єдиного візуального стилю гри для всіх елементів інтерфейсу та оточення;
- забезпечення читабельності тексту, іконок та показників інтерфейсу;
- відображення відкритого ігрового світу з можливістю вільного пересування персонажа;
- відображення головного меню гри з можливістю почати нову гру, продовжити гру та вийти з гри;
- відображення відкритого ігрового світу з можливістю вільного пересування персонажа та дослідження території;
- інтерфейс керування персонажем (HUD): відображення здоров'я, спраги, голоду та інвентарю;
- відображення та керування інвентарем гравця з можливістю перетягування предметів;

- інтерфейс системи квестів: перегляд активних і виконаних завдань, опис квестів та нагород;
- інтерфейс діалогів з NPC з можливістю вибору варіантів відповіді.

1.5.2 Нефункціональні вимоги

Для забезпечення оптимальної якості ігрового досвіду та стабільної роботи на цільових платформах встановлено такі вимоги:

- сумісність з операційною системою Windows;
- забезпечення стабільного ігрового процесу з частотою кадрів 40-50к/с;
- мінімальні системні вимоги: Процесор: INTEL CORE I5-8400 або AMD RYZEN 3 3300 X RAM: 6 гб Відеокарта: NVIDIA GeForce GTX 1060 3GB або AMD Radeon RX 570 4GB;
- рекомендовані системні вимоги: Процесор: INTEL CORE I5-8700 чи AMD RYZEN 5 3600 X RAM: 8 гб Відеокарта: NVIDIA GeForce GTX 1080Ti or AMD Radeon RX 5700 XT.

2 ЗАСОБИ РЕАЛІЗАЦІЇ FRONTEND-ЧАСТИНИ ВІДЕОГРИ

2.1 Мова програмування C#

Розроблена корпорацією Microsoft, мова C# поєднує в собі потужність об'єктно-орієнтованого підходу та гнучкість у створенні цифрових продуктів будь-якої складності. Сьогодні вона є галузевим стандартом у геймдеві завдяки рушію Unity, що дозволяє ефективно реалізовувати проекти у двовимірному та тривимірному просторі.

Популярність C# серед геймдев-розробників зумовлена вдалим поєднанням високої продуктивності із сучасним та зрозумілим синтаксисом. Завдяки багатому інструментарію мова дозволяє ефективно реалізовувати складну ігрову логіку, системи взаємодії об'єктів та інтерфейси. Важливою перевагою є також автоматизація керування пам'яттю, що спрощує процес програмування та мінімізує ризик виникнення технічних помилок.

Характерною рисою цієї мови є забезпечення високого рівня стабільності та надійності при створенні програмних рішень. Завдяки широкому вибору документації та великій спільноті розробників, програмісти можуть швидко знаходити рішення для реалізації необхідного функціоналу [14–15]. Даний аспект є критичним у геймдеві, особливо коли йдеться про архітектуру ігрових систем чи вдосконалення механік взаємодії.

2.1.1 Переваги та недоліки C#

Основні переваги та недоліки C# як інструменту розробки ігор зведені до таблиці 2.1. Це дозволяє краще оцінити сильні сторони мови, а також врахувати можливі обмеження під час створення ігрового проєкту [14–15].

Таблиця 2.1

Переваги та недоліки C#

Переваги C#	Недоліки C#
Простий та зрозумілий синтаксис, що полегшує написання та підтримку програмного коду.	Нижча продуктивність у порівнянні з мовами низького рівня, зокрема C++.
Підтримка принципів об'єктно-орієнтованого програмування, що спрощує модульність і масштабування проєкту.	Залежність від середовища виконання .NET, що може збільшувати використання системних ресурсів.
Тісна інтеграція з рушієм Unity, який використовує C# як основну мову програмування.	У великих проєктах можливе підвищене навантаження на пам'ять та процесор.
Можливість швидкої реалізації ігрових механік, анімацій та взаємодії персонажа з оточенням.	Автоматичне очищення пам'яті іноді викликає короткочасні затримки у роботі гри.
Велика кількість готових бібліотек та інструментів для прискорення процесу розробки.	Необхідність додаткового контролю використання ресурсів для оптимізації продуктивності.
Автоматичне керування пам'яттю за допомогою збирача сміття, що зменшує ризик витоків пам'яті.	Деякі низькорівневі операції реалізуються складніше, ніж у мовах системного програмування.
Підтримка кросплатформної розробки для Windows, Android, iOS та інших платформ.	Обмежені можливості глибокої оптимізації окремих компонентів гри.
Скорочення часу розробки та спрощення тестування програмного продукту.	

Підсумовуючи, можна стверджувати, що C# є оптимальним вибором для Unity-розробки. Мова забезпечує високу швидкість написання коду та ефективну реалізацію механік, поєднуючи зручність високорівневого синтаксису з широкими можливостями сучасного програмування.

Незважаючи на певні нюанси з керуванням пам'яттю та продуктивністю, переваги C# для більшості ігрових проєктів залишаються незаперечними. Ця мова дозволяє пришвидшити реліз, спростити процес налагодження та гарантувати стабільність коду. З огляду на це, C# є оптимальним інструментом для розробки відеогри у жанрі виживання з елементами рольової гри.

2.2 Огляд рушія Unity у контексті розробки інтерфейсу

Unity являє собою один із багатьох відомих ігрових рушіїв, що використовується для створення відеоігор. Платформа дозволяє створювати 2D- та 3D-проєкти будь-якої складності, включно з мобільними застосунками та багатокористувацькими системами. Завдяки широкому інструментарію рушій оптимізує розробку складних геймплейних механік та проектування інтерфейсів. Зокрема, можливості Unity дозволяють створювати інтуїтивно зрозумілий UI, що забезпечує позитивний досвід користувача (UX) та комфортну взаємодію з грою.

Інтерфейс Unity побудовано за допомогою внутрішньої інтегрованої системи GUI, яка дозволяє розробникам створювати різноманітні види меню, кнопок, панелей, тексту, слайдерів та навіть інших типів графічного інтерфейсу. Завдяки можливостям цього рушія користувачі можуть самостійно кастомізувати інтерфейс меню. Такий функціонал значно полегшує процес розробки та адаптації UI під монітори з різною роздільною здатністю [16-17].

Що стосується анімації, обробки користувацьких подій та можливості інтеграції графічного інтерфейсу гравця з іншими аспектами гри, Unity надає розробникам багато інструментів для створення візуально гарного та інтуїтивно зрозумілого інтерфейсу користувача для своєї гри.

2.2.1 Характеристика можливостей рушія Unity у реалізації frontend-частини

Розробка на Unity відкриває широкі можливості для роботи з візуалом у будь-якій стилістиці. Основним інструментом є система Unity UI, яка дозволяє створювати графічні елементи інтерфейсу без необхідності використання сторонніх програмних засобів. А вбудований магазин ассетів, дозволяє легко та без витрати часу знайти потрібні моделі чи спрайти.

Можливості Unity дають змогу реалізувати адаптивний інтерфейс, що коректно відображається на різних пристроях. Використання інтерактивних меню, текстових блоків та кнопок у поєднанні з динамічними анімаціями переходів робить взаємодію з грою інтуїтивно зрозумілою, забезпечуючи комфортний і приємний ігровий досвід.

Зручна система створення інтерфейсів у Unity робить процес розробки гри набагато комфортнішим і швидшим. Кожен UI-елемент може містити окремі компоненти, які відповідають за відображення, анімацію або взаємодію з користувачем [17]. Завдяки цьому розробник може швидше редагувати та впроваджувати готові рішення в різних частинах програми.

До того ж, рушій Unity використовує спеціальну систему Canvas, яка автоматично розміщує елементи інтерфейсу поверх ігрового світу та забезпечує їхнє правильне відображення на екрані [18]. То ж, цей ігровий рушій не лише спрощує створення зручного та функціонального ігрового інтерфейсу, але й значно знижує навантаження на систему, оптимізуючи роботу всієї візуальної частини гри.

2.2.2 Переваги та недоліки рушія Unity у frontend-розробці

Головна перевага з усіх в Unity, це зручний візуальний редактор інтерфейсів, за допомогою якого можна швидко створювати і перевіряти UI-елементи. Це дозволяє легко редагувати UI, не зачіпаючи внутрішню логіку коду.

Другою перевагою є підтримка кросплатформності. Інтерфейс, створений у Unity, може працювати на різних платформах без суттєвих змін у структурі проєкту

[16]. Також рушій має велику кількість документації, готових ресурсів та активну спільноту, що значно полегшує процес створення frontend-частини гри.

Рушій забезпечує інтеграцію інтерфейсу з іншими системами гри, зокрема анімацією, аудіосистемою та ігровою логікою [17]. Це дозволяє створювати інтерактивний та динамічний користувацький інтерфейс, який дуже впливає на сприйняття атмосфери гравця.

Серед слабкостей Unity можна виділити підвищене використання системних ресурсів, коли працює зі складними елементами інтерфейсу користувача. Перевантаженість інтерфейсу та анімацій може негативно впливати на продуктивність гри, особливо на слабких пристроях. Ще й при неправильній організації Canvas та UI-компонентів можуть виникати труднощі з оптимізацією та адаптацією інтерфейсу до різних роздільних здатностей екрана [16-17].

Іноді, для реалізації складних або нестандартних елементів інтерфейсу виникає потреба у використанні додаткових плагінів або власних програмних рішень, що може ускладнювати процес розробки.

2.2.3 Особливості використання мови програмування C# при розробці frontend-частини в Unity

В Unity мова програмування C# дуже часто використовується для створення логіки роботи інтерфейсу та взаємодії користувача з елементами UI. За допомогою скриптів відбувається обробка натискань кнопок, оновлення текстових даних, відображення ігрових показників та керування меню. C# працює за принципом компонентної системи, де скрипти прикріплюються до об'єктів сцени та керують їхньою поведінкою [17].

Використання мови C# дозволяє динамічно оновлювати елементи інтерфейсу безпосередньо під час ігрового процесу. Це забезпечує актуалізацію текстів, зображень, слайдерів стану персонажа та інших показників у реальному часі. Крім того, програмна логіка відповідає за роботу системи меню, перемикання між сценами та безшовну інтеграцію frontend-частини з іншими модулями гри [17-18].

Завдяки зв'язці C# та Unity розробники отримують зрозумілу структуру проєкту, яку легко доповнювати новими функціями. Завдяки цьому забезпечується стабільна робота фронтенду та зручна взаємодія користувача з відеогрою.

2.3 Використання середовища розробки Visual Studio у поєднанні з рушієм Unity

Серед розмаїття інструментів для створення ПЗ, Visual Studio виділяється як основне рішення для реалізації ігрових проєктів у екосистемі Unity. VS забезпечує зручні інструменти для написання, зміни та виправлення програмного коду на мові C#. Під час створення відеоігор, саме це середовище застосовується для написання скриптів, реалізації ігрової логіки а роботи з візуальною частиною проєкту. Тісна взаємодія з Unity забезпечує можливість миттєвого редагування коду, що дозволяє проводити комплексну апробацію механік та ідентифікувати програмні недоліки безпосередньо на етапі створення продукту.

VS пропонує широкий спектр можливостей, включаючи автоматичне доповнення коду, підсвічування синтаксису та інструменти для налагодження, що значно підвищує ефективність роботи програміста. Впровадження цього інструменту виступає каталізатором ефективності програміста, трансформуючи складні етапи девелопменту в прозорий та контрольований процес.

Комбінація Unity та Visual Studio є найліпшим рішенням для створення сучасних відеоігор, оскільки дане середовище забезпечує стабільну роботу з масштабними проєктами та відповідає популярним підходам у створенні ПЗ. Завдяки цьому розробник може ефективно організувати структуру коду та оптимізувати всі етапи розробки гри.

2.3.1 Характеристика середовища розробки Visual Studio

Visual Studio є інтегрованим середовищем розробки, який використовується для створення та проєктування програмних застосунків різного типу та

призначення. Воно підтримує роботу з багатьма мовами програмування, однак у поєднанні з Unity найчастіше використовується саме мова C# [20].

Середовище має зручний інтерфейс та широкий набір інструментів для роботи з програмним кодом. Ключовою фішкою Visual Studio є функція IntelliSense. Ця інтелектуальна система автодоповнення не лише пришвидшує написання коду, а й працює як запобіжник, мінімізуючи ймовірність появи синтаксичних помилок [21]. Це середовище підтримує функцію налагодження програмного коду в режимі реального часу. Розробник може встановлювати точки зупинки, перевіряти значення змінних та аналізувати виконання програми, що значно спрощує пошук і виправлення помилок.

Серед інших переваг варто виділити інструменти для логічної організації проєкту та швидкого переходу між його компонентами. Використання додаткових інструментів стає ключовим фактором при менеджменті великих проєктів зі складними ієрархіями скриптів. Модульність середовища дозволяє кастомізувати робочий простір за допомогою зовнішніх ресурсів, оптимізуючи кожен етап створення гри та покращуючи ергономіку розробки.

Якщо підсумовуючи, то середовище розробки має вбудовані засоби аналізу коду, які допомагають виявляти потенційні помилки та покращувати якість ПЗ [21]. Завдяки цьому команда може працювати в єдиному стилі, що значно пришвидшує внесення правок та розвиток проєкту в майбутньому.

2.3.2 Інтеграція Visual Studio з рушієм Unity

Інтеграція Visual Studio з Unity забезпечує зручну взаємодію під час процесу розробки. Рушій сам готує файли для VS, що дозволяє редагувати код, не виходячи з робочого простору розробки. Під час спільної роботи цих двох продуктів, програміст має доступ до всіх класів, методів та бібліотек рушія. Такий підхід значно пришвидшує написання та коригування скриптів, дозволяючи повноцінно використовувати інструментарій Unity для програмування ігрових механік та візуального інтерфейсу [20].

Важливою перевагою середовища є можливість налагодження гри безпосередньо під час її запуску в Unity [20-21]. Інструментарій Visual Studio дозволяє здійснювати оперативний аналіз коду, тестування елементів UI та налагодження механік безпосередньо під час виконання. Функція автоматичного оновлення змінених скриптів оптимізує робочий процес, суттєво скорочуючи час розробки та підвищуючи загальну стабільність програмного коду гри.

Розробники також відділяють зручну підтримку синхронізації проєкту та автоматичне підключення необхідних бібліотек Unity до середовища розробки [20]. Це суттєво полегшує конфігурацію середовища розробки та гарантує стабільне виконання програмного коду.

Інтеграція Unity з Visual Studio формує оптимізований пайплайн розробки, що базується на сучасних стандартах програмування. Таке поєднання дозволяє оптимізувати процес створення frontend-частини гри та забезпечити стабільну роботу програмного продукту.

3 ПРОЄКТУВАННЯ ТА РОЗРОБКА FRONTEND-ЧАСТИНИ ВІДЕОГРИ

3.1 Розробка візуального дизайну та стилю гри

Початок проєкту розпочався з визначення візуального стилю відеогри. Оскільки за сюжетом події відбуваються у всесвіті тварин, стиль необхідно було обрати таким, що відповідає загальній атмосфері гри. Після аналізу аналогів і певних міркувань було обрано мультяшний та яскравий стиль. Саме він найкраще передає настрій гри, адже реалістичний стиль виглядав надто деталізованим, серйозним і навіть дещо моторошним у контексті світу тварин. Піксельний стиль також було відхилено, оскільки деякі візуальні рішення могли б бути неінтуїтивно зрозумілими для гравця під час ігрового процесу.

Після визначення стилю основне завдання полягало у підборі відповідних ассетів, моделей, предметів та інших візуальних об'єктів, що гармонійно вписуються в обрану стилістику. У першу чергу увагу було зосереджено на пошуку моделі головного героя – бобра. Далі було підібрано всі необхідні елементи для створення ландшафту: траву, квіти, текстури землі, піску та снігу, а також дерева і каміння. Окрему увагу приділено моделям будівель і споруд, оскільки за задумом карта містить кілька локацій, доступних для дослідження гравцем.

Усі відібрані елементи були скориговані відповідно до художнього стилю гри для забезпечення їх гармонійного поєднання. Це дозволило підготувати ресурси до взаємодії з користувачем, сценарії якої представлені на загальній діаграмі варіантів використання (рис. 3.1). Завершенням цього етапу став імпорт підготовлених моделей та текстур у Unity для подальшого налаштування ігрового середовища.

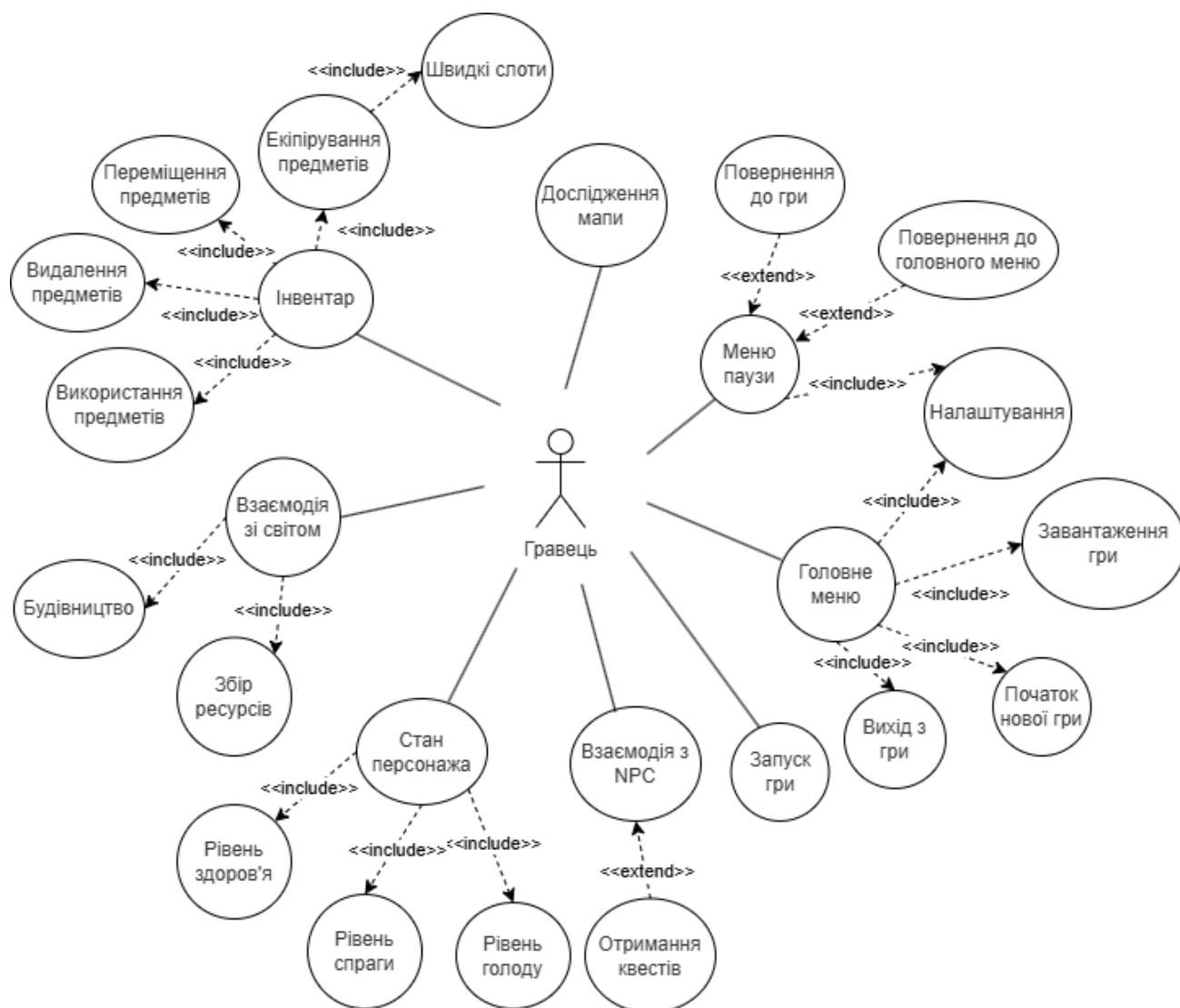


Рис 3.1 UML-діаграма варіантів використання frontend-частини відеогри

3.2 Проєктування ігрового світу (мапи)

Розробка мапи почалася зі створення базової основи для майбутньої мапи світу, де саме будуть розташовані всі ключові локації, ресурси та NPC. Цей етап складався з кількох послідовних кроків:

- створення початкової платформи;
- надання випуклості ландшафту;
- надання текстур до карти.



Рис 3.2 Вигляд мапи згори

Після створення базової структури було розміщено основні об'єкти ігрового середовища – дерева, каміння та гілки, адже вони є одним із головних ресурсів для виживання. Для надання мапі більш привабливого та живого вигляду додано елементи флори: різні види квітів і траву, що формують природне оточення. Також для більшої краси та збереження мультяшного стилю окремі гірські ділянки були вкриті снігом, а деякі частини мапи отримали піщані текстури. Це дозволило урізноманітнити ландшафт і надати йому більш унікального вигляду.



Рис 3.3 Вигляд мапи від лиця персонажа

На цьому етапі розробка карти вийшла на фінальну стадію: сформовано чітку структуру та досягнуто необхідного рівня деталізації, що дозволяє перейти до наповнення контентом і фінальної оптимізації.

3.2.1 Містечко з NPC

Головною локацією гри стало невелике містечко, у якому проживають неігрові персонажі, адже саме тут гравець взаємодіє зі світом найбільш активно. У межах цієї локації він має змогу отримувати квести, просуватися по сюжетній лінії та поступово дізнаватися історію і особливості ігрового всесвіту. Таким чином, містечко виконує не лише декоративну, а й важливу функціональну роль у геймплеї. Основу забудови складають невеликі будиночки, оформлені у відповідній стилістиці, які гармонійно вписуються в загальну атмосферу гри. Кожна споруда доповнена різноманітними деталями, що імітують повсякденне життя мешканців. По серед міста було розташовано колодязь, що додає особливого шарму та родзинки місцевості. Детальніше як це виглядає можна побачити на рис 3.4.



Рис 3.4 Містечко NPC

Неігрові персонажі, так само як і головний герой, представлені у вигляді антропоморфних тварин, тому саме вони населяють ігрове містечко. Такий підхід дозволяє зберегти цілісність світу та підкреслити обрану стилістику гри. Саме неігрові персонажі виступають основними рушіями розвитку сюжету, оскільки через взаємодію з ними гравець отримує ключову інформацію про світ, завдання та подальший розвиток подій.



Рис 3.5 NPC койот

3.2.2 Ферма

Напроти містечка розташована ферма, яка відіграє важливу роль у житті місцевих жителів, забезпечуючи їх основними ресурсами для виживання та розвитку. Саме тут вони мають змогу вирощувати різноманітні сільськогосподарські культури, що використовуються як для власного споживання, так і для подальшого обміну чи торгівлі. Територія ферми включає кілька амбарів, призначених для зберігання сіна та інших зібраних ресурсів. Поруч розташований вітряний млин, який є важливим функціональним об'єктом – саме тут жителі переробляють зібране зерно на борошно, необхідне для подальшого використання.

Центральним елементом ферми виступають поля, де вирощуються різні культури, зокрема пшениця, морква та буряк.



Рис 3.6 Вигляд ферми

3.2.3 Цвинтар

Ворожою локацією у грі було обрано цвинтар, що природно створює напружену та дещо моторошну атмосферу. Саме тут мешкають тварини-вороги, які атакують гравця та створюють додаткові виклики під час проходження. Основними противниками стали змії, що оселилися на цій похмурій території та органічно вписуються в загальний настрій локації.

Цвинтар складається з центральної споруди – склепу, оточений численними надгробками різних форм і розмірів. Для підсилення страхітливої атмосфери було додано труну зі скелетом, а також засохлі дерева, на яких розміщено старі ліхтарі. Додатково, для надання атмосфері легкого гелловінського настрою, територію було прикрашено гарбузами та свічками. Ці елементи не лише урізноманітнюють візуальне оформлення, але й додають контрасту між мультяшним стилем гри та моторошною тематикою самої локації.



Рис 3.7 Цвинтар

3.2.4 Закинутий табір

Якщо гравець почне досліджувати мапу, він зможе знайти ще одну локацію – Закинутий табір. Він складається з декілька палаток, біля яких розташувалися деякі закинуті речі. Посередині, як це зазвичай буває у кемпінгах, знаходиться погашене багаття, а навколо розкидані невикористані дрова. Атмосферу доповнюють сліди тривалого перебування мешканців, що створює відчуття покинутості та загадковості. Така локація відтворює самотню атмосферу і додає грі більшої сюжетності та занурення в ігровий світ.



Рис 3.8 Закинутий табір

3.3 Розробка меню

В межах розробки було впроваджено комплексний інтерфейс користувача, що гарантує безперерйну взаємодію з грою як під час запуску, так і в режимі реального часу. Меню є не невід’ємною частиною UI, оскільки саме через нього гравець отримує доступ до основних функцій: початку нової гри, налаштувань, повернення до головного меню, а також керування грою в процесі проходження.

Система меню була розділена на дві логічні частини: головне меню, яке відображається перед початком гри, та внутрішньоігрове меню паузи. Для кожної з цих частин було створено окремі програмні класи, що дозволяє чітко розмежувати побудову між компонентами та зробити код більш структурованим і зручним для подальшого розширення. Всю архітектуру системи меню можна відобразити на діаграмі класів, рисунок 3.9.

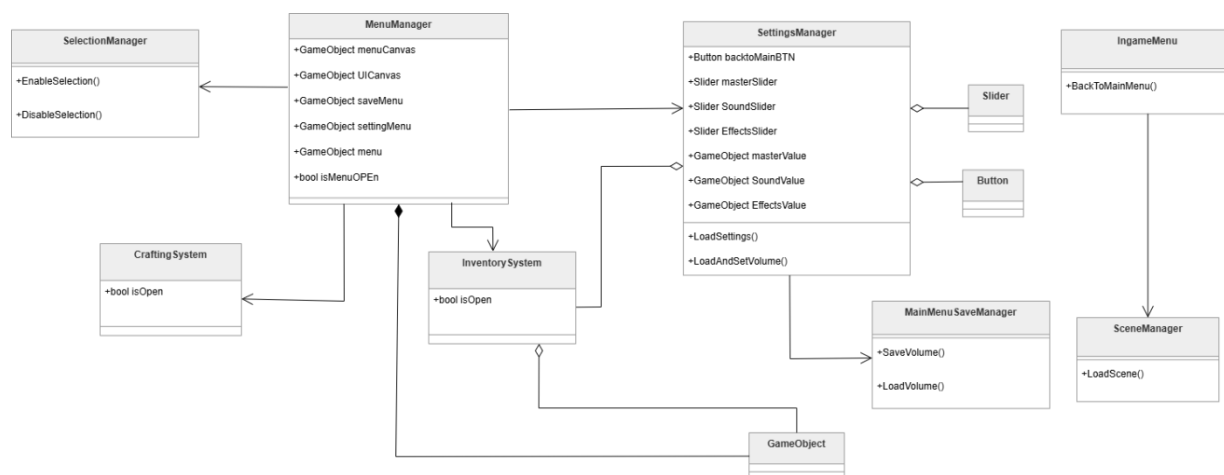


Рис. 3.9 Діаграма класів системи меню

3.3.1 Головне меню

Головне меню – це перший крок у знайомстві з грою, тож його реалізація має бути надзвичайно простою і зрозумілою. Для керування логікою головного меню було створено клас **MainMenuController**, який відповідає за обробку дій користувача при натисканні кнопок інтерфейсу (див. Рис. 3.10). Найголовнішим

методом в цьому класі є `StartNewGame()`, що відповідає за завантаження основної ігрової сцени. Виконання цього завантаження відбувається за допомогою вбудованої системи керування сценами, що дозволяє здійснити плавний перехід від меню до безпосереднього ігрового процесу. Отже, натиснувши кнопку початку гри користувач одразу переходить у створений ігровий світ. Метод `ExitGame()` відповідає за завершення роботи програми.

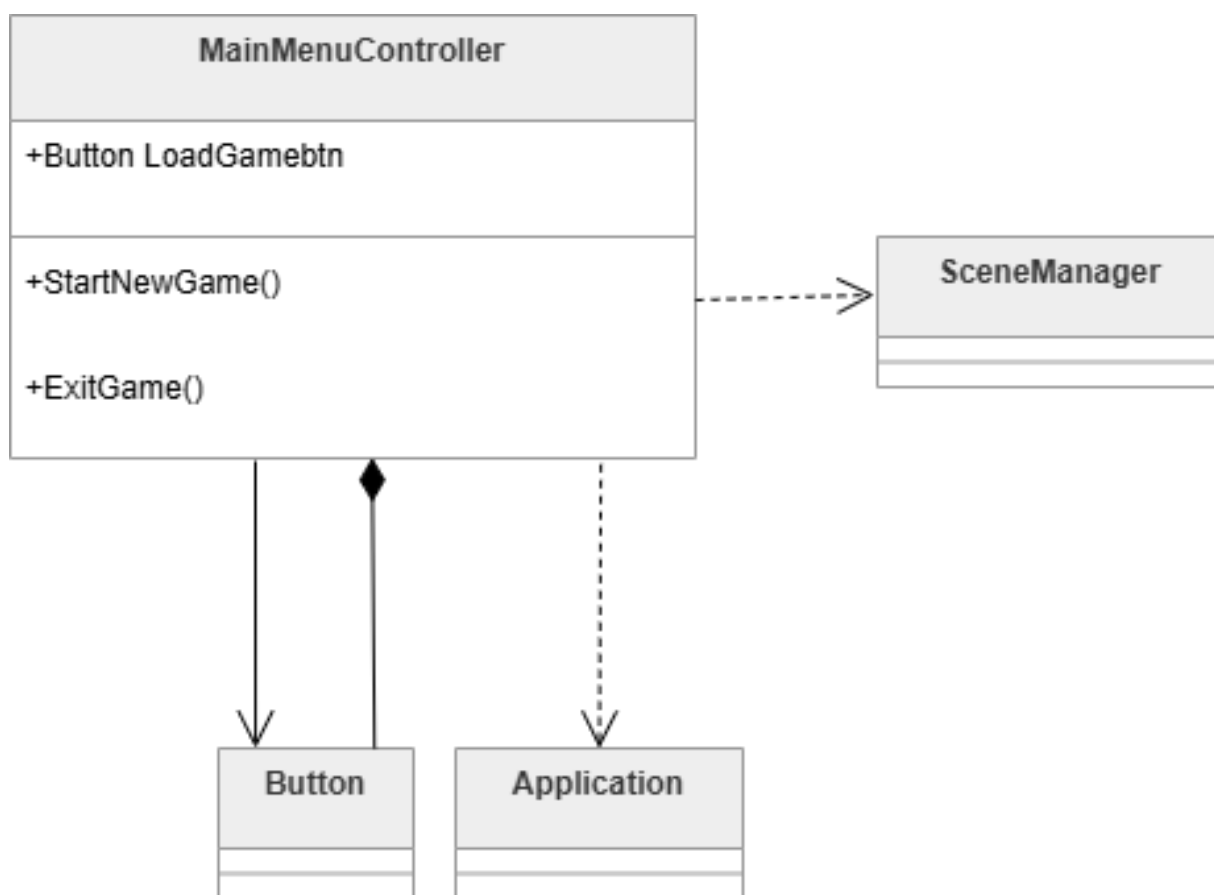


Рис. 3.10 Діаграма класу `MainMenuController`

Одним з ключових елементів головного меню є система налаштувань, яка представлена класом `SettingsManager`. Цей клас реалізований за принципом `singleton`, що передбачає існування єдиного екземпляра менеджера налаштувань на протязі всієї сесії гри. Такий підхід дозволяє централізовано управляти параметрами та виключати можливі конфлікти між різними модулями застосунку.

У методі Awake() відбувається перевірка, чи є вже створений екземпляр, і в разі повторного створення видаляється зайвий об'єкт.

Метод Start() відповідає за ініціалізацію взаємодії з інтерфейсом. Зокрема, для кнопки повернення в меню встановлюється обробник події, який при натисканні зберігає поточні налаштування звуків. Збереження здійснюється через окремий менеджер, що дає можливість записувати рівень гучності та відновлювати його при наступному запуску гри.

Ще в методі Start() запускається співклас LoadSettings(). Використання співкласу дає можливість завантажити налаштування з невеликою затримкою, що унеможливорює появу помилок при ініціалізації UI. Він викликає функцію LoadAndSetVolume(), яка отримує збережені значення параметрів звуку та встановлює їх у відповідні елементи інтерфейсу, слайдери. Це дозволяє відновити попередній стан налаштувань і підвищує комфорт гравця.

У методі Update() відбувається постійне оновлення текстових значень, що відображають рівень гучності. Для цього кожне значення слайдера конвертується у рядок та відображається через компонент TextMeshProUGUI.

Візуальне оформлення інтерфейсу головного меню відповідає загальному художньому стилю гри (Cartoon Style). Графічна композиція складається з пейзажного бекграунду та центральної панелі у формі дерев'яної плашки.

У верхній частині екрана розміщено назву гри, виконану стилізованим шрифтом. Центральна частина інтерфейсу містить панель із кнопками, яка оформлена у вигляді дерев'яної таблички. Кнопки меню: New Game, Load Game, Settings та Exit, розроблені так щоб не зливатися з фоном. Обрана вертикальна ієрархія пунктів меню забезпечує інтуїтивну навігацію користувача.



Рис. 3.11 Вигляд Головного меню

3.3.2 Меню паузи

Меню паузи також основна частина геймплею, адже воно дозволяє тимчасово призупинити гру та отримати доступ до додаткових функцій без повернення до головного меню.

Його реалізація здійснена у класі `MenuManager`, який відповідає за відкриття, закриття та керування станами внутрішньоігрового меню. Як і в інших менеджерів, цей клас реалізований за принципом `singleton`. У методі `Awake()` відбувається перевірка існування екземпляра, що забезпечує централізоване керування меню.

Основна логіка роботи меню реалізована у методі `Update()`, де постійно відслідковується натискання клавіші відкриття меню. При першому натисканні виконується активація меню паузи: основний ігровий інтерфейс вимикається, а меню відображається на екрані. Одночасно змінюється і поведінка курсора, він стає видимим і розблокованим, що дозволяє взаємодіяти з кнопками та іншими елементами інтерфейсу.

Окрім цього при відкритті меню вимикається система вибору об'єктів у грі, реалізація відбувається через звернення до відповідного менеджера, що дає змогу уникнути небажаної взаємодії з ігровим середовищем під час перебування в меню.

При повторному натисканні клавіші виконується закриття меню. При цьому всі додаткові вкладки, такі як меню налаштувань або збереження, скидаються до початкового стану, а головне меню паузи стає активним. Відновлюється основний інтерфейс гри, що дає змогу користувачу продовжити гру.

Особливу увагу приділено керуванню курсором. Якщо під час закриття меню не активні інші системи, такі як інвентар або крафтинг, курсор блокується та стає невидимим. Це забезпечує повернення до стандартного режиму керування персонажем.

Для реалізації переходу до головного меню використовується окремий клас `IngameMenu`. Його метод `BackToMainMenu()` відповідає за завантаження сцени головного меню, що дозволяє користувачу в будь-який момент вийти з поточної гри.

Дизайн меню паузи успадковує стилістику головного екрана: воно представлене у формі пергаментної панелі з вертикальним списком кнопок (`Save Game`, `Settings`, `Exit`). Кожна кнопка реалізована як об'ємний об'єкт на темному підкладі. Використання чіткого контрастного шрифту гарантує відмінну читабельність інтерфейсу незалежно від ігрової сцени на фоні.



Рис. 3.12 Візуалізація меню паузи

3.4 Реалізація системи інвентарю гравця

Система інвентарю є однією з ключових складових ігрового процесу у відеогрі жанру виживання, через яку гравець взаємодіє з різними предметами, ресурсами та елементами спорядження. Система інвентарю відповідає за зберігання предметів, їх використання, перенесення між слотами, екіпірування, видалення та взаємодію з іншими внутрішньо-ігровими механіками. Її реалізація базується на наборі взаємопов'язаних класів, де кожен виконує свою функцію в архітектурі системи. Візуальне представлення системи інвентаризації наведено на UML-діаграмі класів (Рис. 3.12).

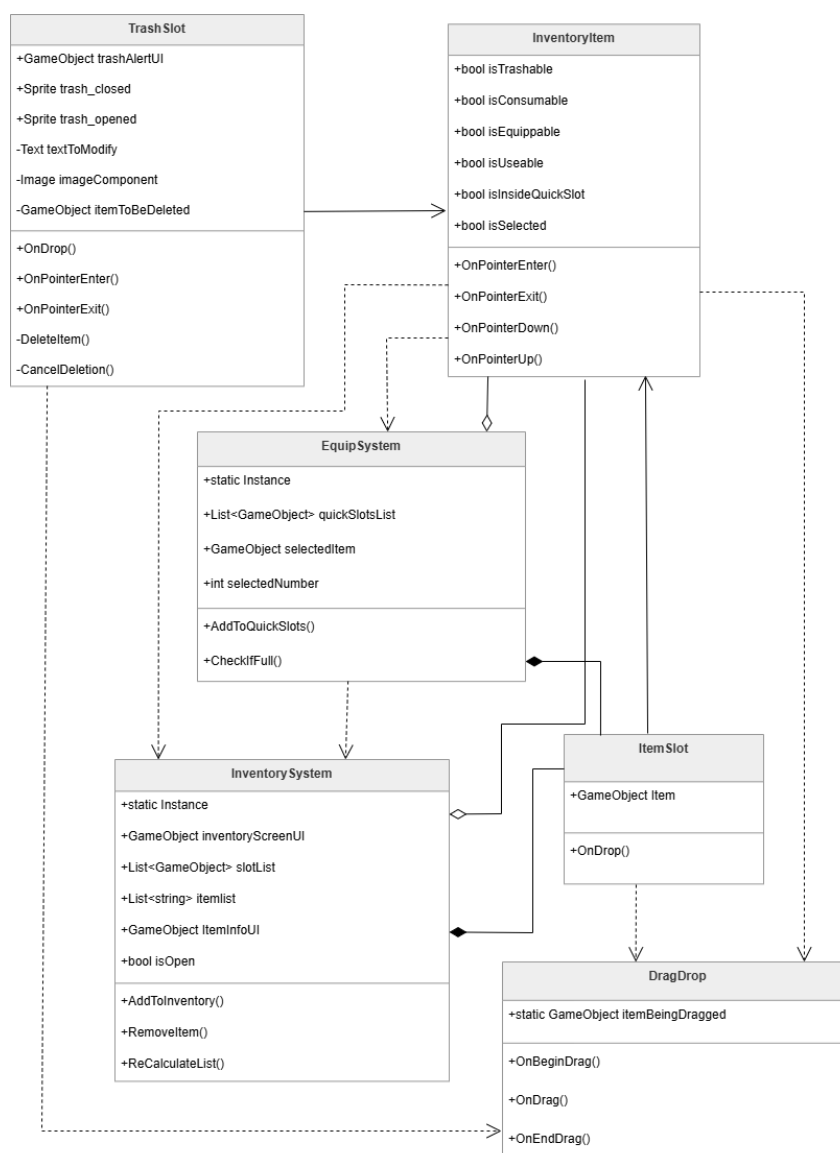


Рис. 3.12 Діаграма класів системи інвентарю

Основою системи є клас `InventorySystem`, який реалізує функціонал керування інвентарем гравця. Цей клас спроектований за принципом `singleton`, що гарантує існування лише одного об'єкта інвентаря протягом часу гри. У методі `Awake()` виконується перевірка, чи був уже створений інший екземпляр класу. Якщо так, то новий екземпляр видаляється, аби не виникало конфлікту та дублікатів. Завдяки цьому досягається унікальність інвентаря та можливість керувати предметами в ньому.

Метод `Start()` відповідає за початкове налаштування системи. Після запуску гри інвентар за замовчуванням знаходиться у закритому стані, а курсор приховується для забезпечення стандартного режиму керування персонажем. Додатково викликається метод `PopulateSlotList()`, який автоматично знаходить усі слоти інвентаря серед дочірніх елементів інтерфейсу та додає їх до спеціального списку. Це дає можливість динамічно працювати зі слотами без необхідності вручну призначати кожен елемент через інспектор `Unity`.

Відкриття та закриття інвентарю реалізована у методі `Update()`. Саме в ній здійснюється відслідковування натискання клавіші `I`. Якщо інвентар закритий, виконується активація відповідного інтерфейсу, курсор стає видимим та розблокованим, а система вибору об'єктів тимчасово вимикається. Це потрібно для того, щоб користувача міг нормально взаємодіяти з елементами інтерфейсу. При повторному натисканні клавіші інвентар закривається, з'являється курсор та повертається до режиму керування персонажем.

Також перевіряється система крафтингу, оскільки при її відкритті курсор повинен залишатися активним. Для додавання предметів до інвентарю використовується метод `AddToInventory(string itemName)`. Під час виклику цього методу виконується пошук наступного вільного слота через функцію `FindNextEmptySlot()`. Після цього необхідний об'єкт завантажується з папки ресурсів за допомогою `Resources.Load`, створюється у сцені та автоматично прикріплюється до знайденого слота. Назва предмета додається до окремого списку предметів інвентарю. Після цього система оновлює внутрішній список предметів та викликає оновлення системи крафтингу.

Система сповіщень про підбір предметів виконуються через метод `TriggerPickUpPop()`, він активує спеціальне вікно повідомлення, у якому відображаються назва предмета та його іконка. Для автоматичного приховування повідомлення використовується співклас `HidePopupAfterDelay()`, який через певний проміжок часу деактивує елемент інтерфейсу. Метод `CheckifFull()` використовується для перевірки заповненості інвентарю. Він підраховує кількість слотів, у яких вже знаходяться предмети, та порівнює отримане значення з максимально допустимою кількістю слотів.

Для видалення предметів реалізовано метод `RemoveItem()`. Він проходить по слотах інвентарю у зворотному порядку та видаляє необхідну кількість предметів із відповідною назвою. Після завершення видалення, виконується оновлення списку предметів та перерахунок необхідних ресурсів для крафтингу. Метод `ReCalculateList()` забезпечує синхронізацію внутрішнього списку предметів із фактичним вмістом слотів інвентарю. Під час виконання методу список очищується та повторно заповнюється назвами предметів, що знаходяться у слотах. Додатково виконується видалення службового суфікса (`Clone`), який автоматично додається `Unity` при створенні об'єктів.

Взаємодія окремих предметів реалізована у класі `InventoryItem`. Даний клас відповідає за поведінку конкретного предмета в інтерфейсі інвентарю. У методі `Start()` відбувається отримання посилань на елементи інформаційного вікна предмета. Методи `OnPointerEnter()` та `OnPointerExit()` реалізують систему відображення інформації про предмет. При наведенні курсора на предмет відкривається інформаційне вікно, у якому показується назва предмета, його опис та функціональне призначення. При відведенні курсора вікно автоматично приховується. Також метод `Update()` виконує перевірку стану вибраного предмета. Якщо предмет екіпірований або активний, компонент перетягування тимчасово вимикається, що запобігає випадковому переміщенню предмета між слотами.

Метод `OnPointerDown()` відповідає за обробку натискання правої кнопки миші по предмету. Залежно від типу предмета виконуються різні дії. Якщо предмет є активний, запускається процедура його використання. Для екіпірованих

предметів виконується додавання до системи швидких слотів. Якщо предмет належить до категорії будівельних елементів, активується система будівництва. Метод `UseItem()` реалізує логіку використання предметів для будівництва. Під час його виконання закриваються всі відкриті інтерфейси, блокується курсор та відновлюється керування персонажем. Після цього через конструкцію `switch` визначається тип використаного предмета та активується відповідний режим будівництва. Наприклад, при використанні фундаменту або стіни система переходить у режим розміщення будівельного об'єкта у світі гри. Метод `OnPointerUp()` завершує процес використання споживаного предмета. Якщо предмет був використаний, він видаляється з інвентарю, а система крафтингу оновлює перелік доступних рецептів.

Окремо реалізовано систему зміни характеристик персонажа після споживання предметів. Метод `consumingFunction()` викликає функції зміни здоров'я, голоду та спраги. Методи `healthEffectCalculation()`, `caloriesEffectCalculation()` та `hydrationEffectCalculation()` перевіряють поточні значення параметрів персонажа та не дозволяють перевищувати максимально допустимі значення.

Для реалізації швидкого доступу до предметів створено клас `EquipSystem`. Даний клас також реалізований за принципом `singleton` та відповідає за систему швидких слотів. У методі `Start()` автоматично формується список слотів швидкого доступу. У методі `Update()` відслідковується натискання цифрових клавіш від 1 до 7, які відповідають за вибір активного слота. Метод `SelectQuickSlot()` виконує основну логіку вибору предмета. Якщо слот містить предмет, система визначає, чи був цей слот вже активним. При виборі нового предмета попередній предмет деактивується, а новий позначається як вибраний. Додатково змінюється колір текстового індикатора номера слота, що дозволяє користувачу бачити активний слот. Якщо гравець повторно натискає ту саму клавішу, предмет знімається з екіпірування.

Метод `SetEquippedModel()` відповідає за створення тривимірної моделі предмета у руках персонажа. Попередня модель видаляється, після чого із ресурсів завантажується новий префаб моделі предмета та прикріплюється до спеціального об'єкта `toolHolder`. Це створює ефект фізичної присутності інструмента або зброї у руках персонажа. Методи `FindNextEmptySlot()` та `CheckIfFull()` забезпечують перевірку доступності слотів швидкого доступу. Вони використовуються при екіпіруванні нових предметів.

Механіка перетягування предметів між слотами створена класом `DragDrop`. Даний клас реалізує інтерфейси `IBeginDragHandler`, `IDragHandler` та `IEndDragHandler`. У методі `Awake()` отримуються посилання на компоненти `RectTransform` та `CanvasGroup`, необхідні для роботи UI-елементів.

Метод `OnBeginDrag()` запускається при початку перетягування предмета. У цей момент предмет стає напівпрозорим, вимикається блокування інтерфейсу, а система запам'ятовує початкову позицію та батьківський елемент. Додатково об'єкт тимчасово переноситься у корінь `Canvas`, що дозволяє коректно відображати його поверх інших елементів інтерфейсу.

Метод `OnDrag()` відповідає за безпосереднє переміщення предмета за курсором миші. Координати елемента постійно оновлюються відповідно до руху курсора. Метод `OnEndDrag()` завершує процес перетягування. Якщо предмет не був поміщений у новий слот, він автоматично повертається на попередню позицію. Після завершення перетягування прозорість та взаємодія з інтерфейсом відновлюються.

Клас `ItemSlot` відповідає за поведінку слотів інвентарю. Метод `OnDrop()` активується при переміщенні предмета у слот. Якщо слот порожній, предмет автоматично прикріплюється до нього. Додатково виконується перевірка типу слота. Якщо предмет переміщується до швидкого слота, встановлюється відповідний прапорець екіпірування. При поверненні предмета до звичайного інвентарю цей прапорець скидається. Також система оновлює список предметів інвентарю.

Для видалення предметів реалізовано клас TrashSlot. Даний клас відповідає за взаємодію зі спеціальним слотом видалення предметів. У методі Start() виконується ініціалізація кнопок підтвердження та скасування видалення, а також отримання необхідних компонентів інтерфейсу.

Метод OnDrop() перевіряє, чи може предмет бути видаленим. Якщо предмет підтримує видалення, запускається співклас notifyBeforeDeletion(), який відкриває діалогове вікно підтвердження. Це дозволяє уникнути випадкового видалення важливих предметів.

Метод DeleteItem() остаточно видаляє предмет з інвентарю, оновлює список предметів та систему крафтингу. Метод CancelDeletion() закриває вікно підтвердження без виконання видалення. Методи OnPointerEnter() та OnPointerExit() реалізують візуальний ефект відкриття та закриття кошика при наведенні предмета.

Візуальне оформленню інвентарю та панелі швидкого доступу, стало дуже великим пріоритетом, так як, саме вони є головними інструментами взаємодії гравця зі світом. Уся система побудована на чіткій сітці однакових слотів, тому гравець може швидко й легко орієнтуватися у своїх ресурсах. Фіксоване розташування елементів і рівномірна структура роблять інтерфейс інтуїтивним та комфортним для сприйняття. Фон слотів виконаний до стилю всієї гри, тобто пергаменту. Для видалення предмету було обрано червоний хрестик, що має зрозумілу логіку для гравця.



Рис. 3.13 Візуалізація інвентарю гравця

Окремо також був розроблений дизайн панелі швидкого доступу, яка розташована в нижній частині екрана. Вона дозволяє миттєво використовувати найнеобхідніші предмети під час гри. Панель виконана у форматі горизонтальної сітки слотів, що забезпечує зручний і швидкий доступ. Кожен слот має цифрове позначення від 1 до 7, яке відповідає клавішам клавіатури й дає змогу швидко перемикаватися між предметами без відкриття повного інвентарю. Візуальну частину було створено до подоби інвентарю, тобто було взято такий дизайн слотів, щоб полегшити розуміння користувача.



Рис. 3.14 Візуалізація панелі швидкого доступу гравця

3.5 Розробка відображення станів персонажа

Як і у багатьох відеоіграх жанру виживання відображення станів персонажа є важливою складовою користувацького інтерфейсу, оскільки саме вона забезпечує гравця інформацією про поточний фізичний стан персонажа та безпосередньо впливає на прийняття рішень під час ігрового процесу. Для реалізації даної функціональності було створено окремі класи, що відповідають за візуалізацію рівня здоров'я, голоду та спраги. Структуру системи відображення показників гравця представлено у вигляді UML-діаграми класів на рисунку 3.15.

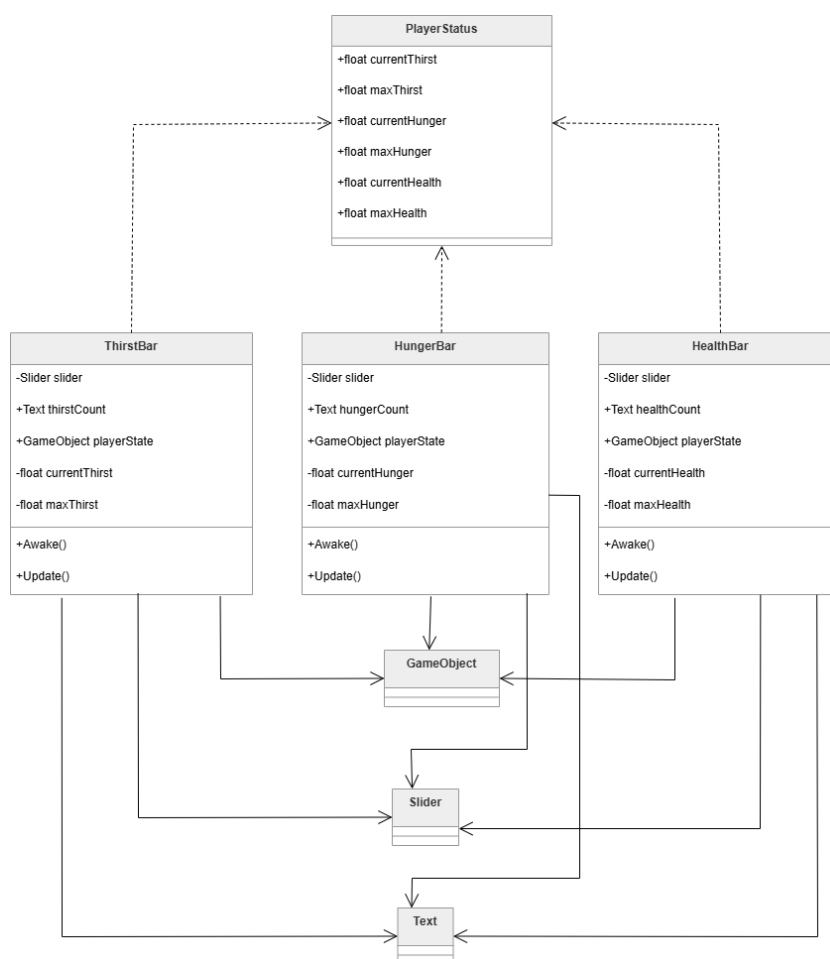


Рис. 3.15 Діаграма класів системи відображення стану гравця

Для реалізації інтерфейсу станів персонажа використано компонент **Slider**, який забезпечує графічне відображення зміни параметрів у режимі реального часу.

Додатково кожен індикатор містить текстове поле, що дозволяє відображати точні числові значення характеристик.

Клас `HealthBar` відповідає за відображення рівня здоров'я персонажа. Основним призначенням цього класу є синхронізація поточного значення здоров'я із відповідним елементом інтерфейсу. У методі `Awake()` виконується отримання посилання на компонент `Slider`, який використовується як основа графічного індикатора.

Основна логіка роботи класу реалізована у методі `Update()`, який виконується кожного кадру. Під час роботи метод отримує поточне та максимальне значення здоров'я із класу `PlayerStatus`. Для цього використовується звернення до компонента, розміщеного на об'єкті персонажа. Після отримання значень виконується обчислення коефіцієнта заповнення шкали. Значення здоров'я ділиться на максимально допустиме значення, у результаті чого формується число у діапазоні від 0 до 1. Отриманий результат передається до компонента `Slider`, що оновлює графічне відображення індикатора. Додатково у методі оновлюється текстовий елемент `healthCount`, який відображає поточний рівень здоров'я у форматі поточне значення чи максимальне значення.

Клас `HungerBar` реалізує систему відображення рівня голоду персонажа. За своєю структурою він подібний до класу здоров'я, однак працює із параметрами ситості. У методі `Awake()` також виконується ініціалізація компонента `Slider`, що відповідає за графічне відображення рівня голоду. Метод `Update()` постійно отримує поточні значення голоду із системи станів персонажа. Після цього виконується обчислення коефіцієнта заповнення шкали аналогічно до системи здоров'я. Отримане значення передається до повзунка, що забезпечує оновлення індикатора. Додатково текстовий компонент `hungerCount` відображає точні числові значення рівня голоду.

Для реалізації системи відображення рівня спраги створено клас `ThirstBar`. Даний клас відповідає за оновлення графічного та текстового представлення рівня водного балансу персонажа. Як і в попередніх системах, у методі `Awake()` виконується отримання посилання на компонент `Slider`, що використовується для

побудови індикатора. У методі Update() система отримує поточний та максимальний рівень спраги із класу PlayerStatus. Після цього розраховується коефіцієнт заповнення шкали, який використовується для оновлення графічного індикатора. Додатково текстовий компонент thirstCount відображає поточне значення у відсотковому форматі.



Рис. 3.16 Графічне відображення станів гравця

Усі слайдери оформлені однаково й компактно розміщені горизонтально. Кожен індикатор має свій колір кольорам, що допомагає гравцю інтуїтивно зрозуміти який слайдер за що відповідає, а числова позначка полегшує надання інформації до користувача, а контраст із фоном, робить їх дуже читабельними.

4 ТЕСТУВАННЯ FRONTEND-ЧАСТИНИ ВІДЕОГРИ

4.1 Тестування ігрового світу (мапи)

Створення мапи завершилося тестуванням, яке підтвердило її технічну стабільність та комфортність ігрового процесу. Основна увага приділялася тестуванню рельєфу місцевості, роботі колізій, відображенню декоративних елементів та загальній продуктивності гри. Оскільки взаємодія зі світом зосереджена в таких точках, як містечко NPC, ферма, цвинтар та покинутий табір, ці локації пройшли окрему фазу тестування як критично важливі для геймплею. Для цього було сформовано набір тестових сценаріїв, результати виконання яких наведено у таблиці 4.1.

Таблиця 4.1

Набір тестових сценаріїв ігрового світу (мапи)

Назва тесту	Опис тестового сценарію	Очікуваний результат	Результат
Тестування пересування персонажа по мапі	Переміщення персонажа на мапі	Персонаж вільно пересувається без застрягань та помилок колізії	Успішно
Тестування рельєфу мапи	Перевірка взаємодії персонажа з рельєфними участками	Персонаж не провалюється під текстури, рух працює коректно	Успішно
Тестування колізій об'єктів	Перевірка взаємодії персонажа з деревами, камінням та спорудами	Колізії працюють правильно, персонаж не проходить крізь об'єкти	Успішно
Тестування декоративних елементів	Перевірка відображення трави, квітів та інших декоративних об'єктів	Усі об'єкти коректно відображаються без графічних дефектів	Успішно

Продовження таблиці 4.1

Набір тестових сценаріїв ігрового світу (мапи)

Назва тесту	Опис тестового сценарію	Очікуваний результат	Результат
Тестування містечка NPC	Переміщення по містечку та взаємодія з основними об'єктами	Всі проходи доступні, взаємодія з NPC працює стабільно	Успішно
Тестування ферми	Дослідження території ферми та проходів між об'єктами	Відсутні невидимі бар'єри та конфлікти моделей	Успішно
Тестування цвинтаря	Перевірка пересування персонажа по цвинтарю та взаємодії з ворогами	Локація працює стабільно, вороги коректно пересуваються	Успішно
Тестування закинутого табору	Перевірка коректності розташування палаток, багаття та декоративних предметів	Усі об'єкти коректно відображаються та не заважають пересуванню персонажа	Успішно
Тестування продуктивності мапи	Перевірка FPS під час перебування у деталізованих локаціях	Відсутні значні просідання продуктивності	Успішно
Тестування завантаження текстур	Перевірка швидкості та коректності завантаження текстур мапи	Текстури завантажуються без затримок і помилок	Успішно

За результатами тестування локації було верифіковано функціональність ключових компонентів. Зокрема, підтверджено належну роботу механік пересування, коректність обробки колізій, а також відповідність візуалізації

текстур та декорацій. А також здійснено перевірку функціонування пріоритетних локацій та аналіз FPS при взаємодії з насиченими об'єктами сценами. Незначні дефекти було ліквідовано на стадії завершальної оптимізації продукту.

4.2 Тестування користувацького інтерфейсу

Користувацький інтерфейс теж був протестований, з метою перевірки стабільності роботи всіх UI-елементів, правильності взаємодії між компонентами та зручності використання для гравця. Приділялося уваги тестуванню головного меню, меню паузи, системи інвентарю, панелі швидкого доступу та системи відображення станів персонажа. Окрім того, було проведено верифікацію функціональності елементів керування (кнопок, слайдерів), механік drag-and-drop, а також коректності візуалізації текстового контенту. З цією метою було розроблено перелік тестових сценаріїв, а отримані під час їх виконання дані систематизовано в таблиці 4.2.

Таблиця 4.2

Набір тестових сценаріїв користувацького інтерфейсу

Назва тесту	Опис тестового сценарію	Очікуваний результат	Результат
Тестування головного меню	Натискання кнопок New Game, Settings та Exit	Кнопки коректно виконують відповідні дії	Успішно
Тестування завантаження сцени	Перевірка переходу з головного меню до ігрової сцени	Ігрова сцена завантажується без помилок	Успішно
Тестування меню паузи	Відкриття та закриття меню під час гри	Меню коректно відкривається та закривається	Успішно
Тестування курсора	Перевірка блокування та відображення курсора	Курсор коректно змінює стан залежно від відкритого меню	Успішно

Продовження таблиці 4.2

Набір тестових сценаріїв користувацького інтерфейсу

Назва тесту	Опис тестового сценарію	Очікуваний результат	Результат
Тестування налаштувань звуку	Зміна значень повзунків гучності	Значення звуку змінюються та зберігаються	Успішно
Тестування інвентарю	Відкриття та закриття інвентарю клавішею I	Інвентар стабільно відкривається та закривається	Успішно
Тестування додавання предметів	Підбір предметів у світі гри	Предмети додаються у вільні слоти інвентарю	Успішно
Тестування перетягування предметів	Переміщення предметів між слотами	Предмети коректно переміщуються між слотами	Успішно
Тестування швидких слотів	Використання клавіш 1-7 для вибору предметів	Предмети коректно екіпіруються	Успішно
Тестування видалення предметів	Переміщення предмета до кошика	Відображається вікно підтвердження видалення	Успішно
Тестування інформації про предмет	Наведення курсора на предмет інвентарю	Відображається назва та опис предмета	Успішно
Тестування використання предметів	Використання їжі або будівельних елементів	Характеристики персонажа змінюються коректно	Успішно
Тестування шкали здоров'я	Отримання персонажем пошкоджень	Індикатор здоров'я оновлюється в реальному часі	Успішно
Тестування шкали голоду	Зменшення рівня ситості персонажа	Шкала голоду змінюється коректно	Успішно
Тестування шкали спраги	Зміна рівня спраги персонажа	Індикатор спраги працює без помилок	Успішно

Продовження таблиці 4.2

Набір тестових сценаріїв користувацького інтерфейсу

Назва тесту	Опис тестового сценарію	Очікуваний результат	Результат
Тестування UI при зміні роздільної здатності	Зміна розміру вікна гри	Інтерфейс коректно масштабується	Успішно
Тестування продуктивності UI	Використання інвентарю та меню під час гри	Інтерфейс працює без затримок і зависань	Успішно

Тестування користувацького інтерфейсу дозволило перевірити правильність роботи всіх основних елементів взаємодії гравця з грою. Разом із цим, підтверджено коректну взаємодію користувача з меню, інвентарем та індикаторами характеристик персонажа. У межах фінального доопрацювання було усунуто виявлені раніше дрібні дефекти UI.

ВИСНОВКИ

1. На основі проведеного дослідження жанру виживання з RPG-елементами визначено його ключові особливості та механіки, проаналізовано роль frontend-складової у забезпеченні взаємодії користувача з ігровими системами, а також досліджено особливості цільової аудиторії й основні вимоги користувачів до інтерфейсу та ігрового досвіду.

2. Проведений аналіз frontend-реалізації у популярних іграх-аналогах допоміг визначити переваги та недоліки різних підходів до організації інтерфейсу користувача, що дозволило визначити функціональні та нефункціональні вимоги до відеогри.

3. Аналіз технічних засобів розробки, дозволив визначити інструменти для розробки, а саме мову програмування C#, ігровий рушій Unity та середовище Visual Studio.

4. У межах проєкту було сформовано візуальний стиль та спроектовано ігровий світ із різними локаціями (містечко NPC, ферма, цвинтар та закинутий табір), а також розроблено архітектуру UI, що охоплює систему меню, інвентар та панелі життєвих показників. На основі розробленої концепції впроваджено функціонал інвентарю, інтегрований із механіками будівництва та крафтингу, разом із тим було реалізовано систему відображення стану персонажа для відстеження його здоров'я, голоду та спраги.

5. Після проведеного тестування ігрового світу та користувацького інтерфейсу, за результатами, критичних помилок не виявлено, а незначні недоліки були усунені.

Робота пройшла апробацію на наукових конференціях, за результатами апробації опубліковано тези доповідей:

Ромашко В.В., Замрій І.В. Архітектура 3D гри в жанрі survival з елементами RPG. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та

рішення в програмній інженерії», 26 листопада 2025 року, Київ, ДУІКТ. Збірник тез. – К.: ДУІКТ, 2025. С.468 -471.

Ромашко В.В., Замрій І.В. Визначення вимог до Frontend-частини 3D гри в жанрі survival з елементами RPG. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, ДУІКТ. Збірник тез. – К.: ДУІКТ, 2026. С.128 -130.

ПЕРЕЛІК ПОСИЛАНЬ

1. Zagal J. Game Design and Role-Playing Games. *Research Gate*. URL: https://www.researchgate.net/publication/331765161_Game_Design_and_Role-Playing_Games. Schell J. The Art of Game Design: A Book of Lenses. 3rd ed. CRC Press, 2020. – 610 с. – ISBN 979-8855589689. (дата звернення: 23.02.2026)
2. Managing and controlling digital role-playing game elements: A current state of affairs / A. De Oliveira Da Rocha Franco та ін. *ScienceDirect*. URL: <https://www.sciencedirect.com/science/article/abs/pii/S1875952124000764>. (дата звернення: 23.02.2026)
3. Koster R. A Theory of Fun for Game Design. 2nd ed. O'Reilly Media, USA: Sebastopol, 2013, 300 p. (дата звернення: 24.02.2026)
4. Caroux L. Presence in video games: A systematic review and meta-analysis of the effects of game design choices. *ScienceDirect*. URL: <https://www.sciencedirect.com/science/article/abs/pii/S0003687022002599>. (дата звернення: 24.02.2026)
5. Friedman A. The Role of Visual Design in Game Design. *Sage Journals*. URL: <https://journals.sagepub.com/doi/abs/10.1177/1555412014559977>. (дата звернення: 25.02.2026)
6. Emotional Design Through Visual Aesthetics in Serious Games: A Scoping Review / L. Kallabis та ін. *IEEE Xplore*. URL: <https://ieeexplore.ieee.org/abstract/document/11202982>. (дата звернення: 25.02.2026)
7. Hardman N. Design Principles Learned From Minecraft. *The BYU Design Review*. URL: <https://www.designreview.byu.edu/collections/design-principles-learned-from-minecraft>. (дата звернення: 01.03.2026)
8. Kaliuzhna T. Cool interface ideas and some problems in The Forest. *Medium*. URL: <https://medium.com/@tetiana.tk.kaliuzhna/cool-interface-ideas-and-some-problems-in-the-forest-e52c754b8277>. (дата звернення: 02.03.2026)
9. Graft K. The Rust Diaries: Letting go of structure in video games. *Game Developer*. URL: <https://www.gamedeveloper.com/design/the-rust-diaries-letting-go-of->

[structure-in-video-games](#). (дата звернення: 03.03.2026)

10. @grazzer88. ARK: Survival Evolved, First Impressions. *digitalvestibule*. URL: <https://digitalvestibule.wordpress.com/2015/06/16/ark-survival-evolved-first-impressions/>. (дата звернення: 04.03.2026)

11. Malecki A. GAME DESIGN ANALYSIS: DON'T STARVE. *Medium*. URL: <https://annamalecki.medium.com/game-design-analysis-dont-starve-50d06561097d>. (дата звернення: 05.03.2026)

12. Функціональні та нефункціональні вимоги (з прикладами). *Visure Solutions*. URL: <https://surli.cc/vchagj>. (дата звернення: 07.03.2026)

13. @Zenva. C# This Tutorial – Complete Guide. *Zenva*. URL: <https://gamedevacademy.org/c-this-tutorial-complete-guide-2/>. (дата звернення: 08.03.2026)

14. Sheldon R. Definition C# (C-Sharp). *TechTarget*. URL: <https://www.techtarget.com/whatis/definition/C-Sharp>. (дата звернення: 08.03.2026)

15. Unity Game Engine: Why Unity Is the Best Game Engine for Your Next Project (2025 Edition). *Maxima Gaming Studio*. URL: <https://maximagamingstudio.com/unity-game-engine-why-unity-is-the-best-game-engine-for-your-next-project-2025-edition/>. (дата звернення: 11.03.2026)

16. Blackman S. Beginning 3D Game Development with Unity 4: All-in-one, Multi-platform Game Development, Berkeley: Apress, 2014, 998 p. (дата звернення: 12.03.2026)

17. Thorn A. Pro Unity Game Development with C#, Berkeley: Apress, 2014, 397 p. (дата звернення: 12.03.2026)

18. Doran J. Unity 2021 Shaders and Effects Cookbook, 4-th ed., Birmingham: Packt, 2021, 738 p. (дата звернення: 13.03.2026)

19. Visual Studio documentation. *Microsoft Documentation*. URL: <https://learn.microsoft.com/en-us/visualstudio/windows/?view=visualstudio>.

20. Visual Studio C# integration. *Unity Manual*. URL: <https://docs.unity.cn/2022.1/Documentation/Manual/VisualStudioIntegration.html>.

21. Change log (Visual Studio Tools for Unity, Windows). *Microsoft*

Documentation.

URL:

<https://learn.microsoft.com/en-us/visualstudio/gamedev/unity/change-log-visual-studio-tools-for-unity>.

ДОДАТОК А ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-
КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



Розробка 3D відеогри "Whispers of the Wildwood" в жанрі виживання з елементами RPG. Спец-частина: розробка frontend-частини системи.

Виконала студентка 4 курсу

групи ПД-43

Ромашко Вероніка Вадимівна

Керівник роботи

завідувач кафедри ІПЗ, док.тех.наук, проф. Замрій Ірина Вікторівна

Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: покращення ігрового досвіду користувача за рахунок поєднання геймплею жанру виживання з ключовими елементами RPG.

Об'єкт дослідження: процес набуття досвіду під час дослідження механік відеогри у жанрі виживання з елементами RPG.

Предмет дослідження: програмні засоби та методи створення геймплею відеогри у жанрі виживання з елементами RPG.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз відеоігор жанру виживання з елементами RPG та визначити його цільову аудиторію.
2. Провести порівняльний аналіз frontend-реалізації у популярних іграх-аналогах.
3. Визначити функціональні та не функціональні вимоги до відеогри.
4. Здійснити аналіз засобів реалізації frontend-частини відеогри.
5. Визначити візуальний дизайн та стиль гри.
6. Спроекувати ігрови світ (мапу): містечко з NPC, ферму, цвинтар та закинутий табір.
7. Розробити головне меню, меню паузи, систему інвентарю та візуалізацію станів персонажа.
8. Провести тестування розроблених механік та мапи.

3

КОНЦЕПТ ГРИ

Сетинг - середньовічне фентезі з розумними тваринами які перейшли до феодального рівня розвитку соціума.

Сюжет - гравець бере на себе роль молодого бобра, який залишився без дому та сім'ї після загадкового катаклізму. Опинившись в незнайомій частині великого лісу, користувач має вижити та знайти відповідь на питання: Що саме сталося з великою плотиною?

Ключові механіки:

- 1) показники виживання (голод, спрага, здоров'я);
- 2) добування ресурсів;
- 3) будівництво бази;
- 4) взаємодія з неігровими персонажами;
- 5) квести.

Ключові особливості:

1. Поєднання жанру виживання з елементами жанру RPG.
2. Заздалегіть створена мапа світу з тематичними локаціями.

4

АНАЛІЗ АНАЛОГІВ

Критерій / Гра	<u>Minecraft</u>	<u>The Forest</u>	<u>Rust</u>	<u>ARK: Survival Evolved</u>	<u>Don't Starve</u>	<u>Whispers of the Wildwood</u>
<u>Мінімалістичний інтерфейс</u>	+	+	-	-	+	+
<u>Висока інформативність</u>	-	-	+	+	+	+
<u>Простота освоєння</u>	+	+	+	+	+	+
<u>Не велика кількість параметрів</u>	+	-	-	-	+	+
<u>Простий UI крафту</u>	+	-	+	+	+	+
<u>Швидкий доступ до предметів</u>	+	-	+	+	-	+
<u>Художня стилізація інтерфейсу</u>	+	+	-	-	+	+
<u>Високий рівень занурення</u>	+	+	-	-	+	+
<u>Не складна структура меню</u>	+	+	-	-	+	+
<u>Не високе когнітивне навантаження</u>	+	+	-	-	+	+
<u>Постійний HUD</u>	+	-	+	+	+	+

5

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Відображення єдиного візуального стилю гри для всіх елементів інтерфейсу та оточення.
2. Забезпечення читабельності тексту, іконок та показників інтерфейсу.
3. Відображення відкритого ігрового світу з можливістю вільного пересування персонажа.
4. Відображення головного меню гри з можливістю почати нову гру, продовжити гру та вийти з гри.
5. Відображення відкритого ігрового світу з можливістю вільного пересування персонажа та дослідження території.
6. Інтерфейс керування персонажем (HUD): відображення здоров'я, спраги, голоду та інвентарю.
7. Відображення та керування інвентарем гравця з можливістю перетягування предметів.
8. Інтерфейс системи квестів: перегляд активних і виконаних завдань, опис квестів та нагород.
9. Інтерфейс діалогів з NPC з можливістю вибору варіантів відповіді.

Нефункціональні вимоги:

1. Сумісність з операційною системою Windows.
2. Забезпечення стабільного ігрового процесу з частотою кадрів 40-50к/с.
3. Мінімальні системні вимоги:
Процесор: INTEL CORE I5-8400 або AMD RYZEN 3 3300X RAM: 6 гб
Відеокарта: NVIDIA GeForce GTX 1060 3GB або AMD Radeon RX 570 4GB.
4. Рекомендовані системні вимоги:
Процесор: INTEL CORE I5-8700 чи AMD RYZEN 5 3600X RAM: 8 гб
Відеокарта: NVIDIA GeForce GTX 1080Ti or AMD Radeon RX 5700 XT.

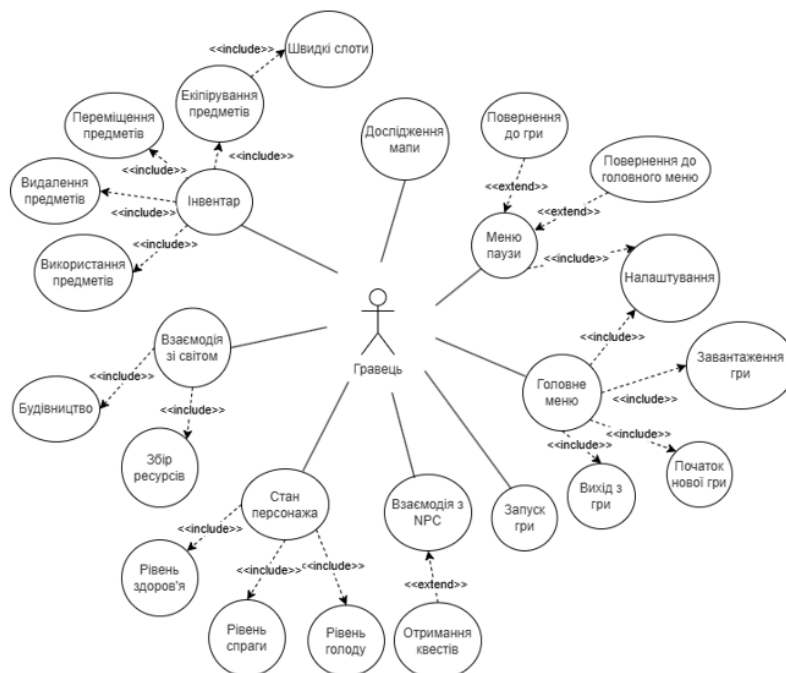
6

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



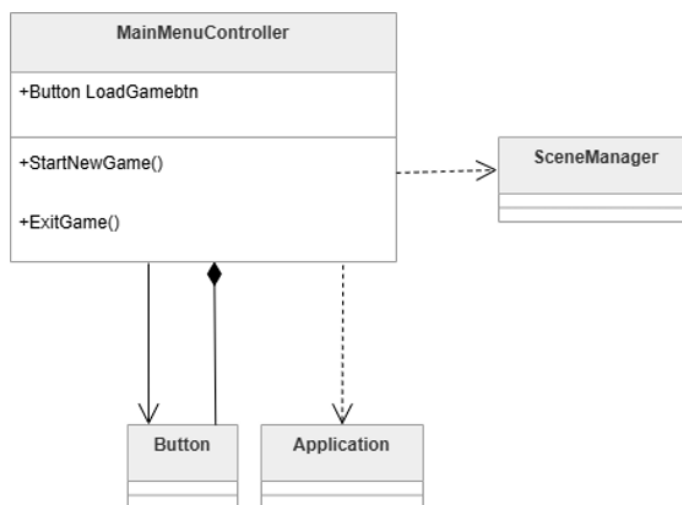
7

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ FRONTEND-ЧАСТИНИ



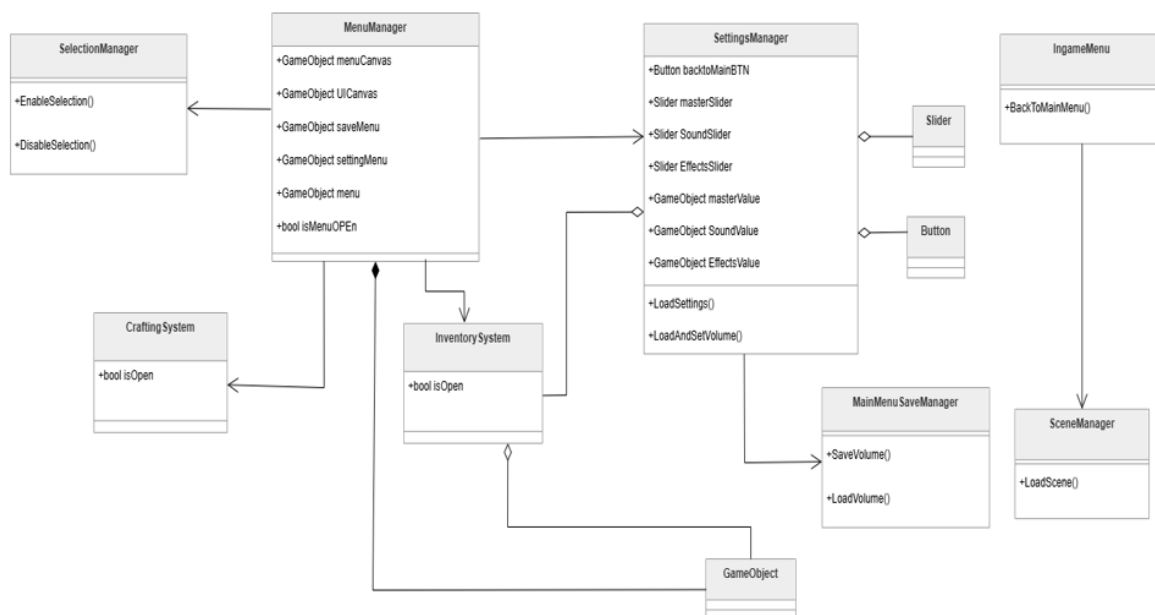
8

ДІАГРАМА КЛАСІВ СИСТЕМИ ГОЛОВНОГО МЕНЮ



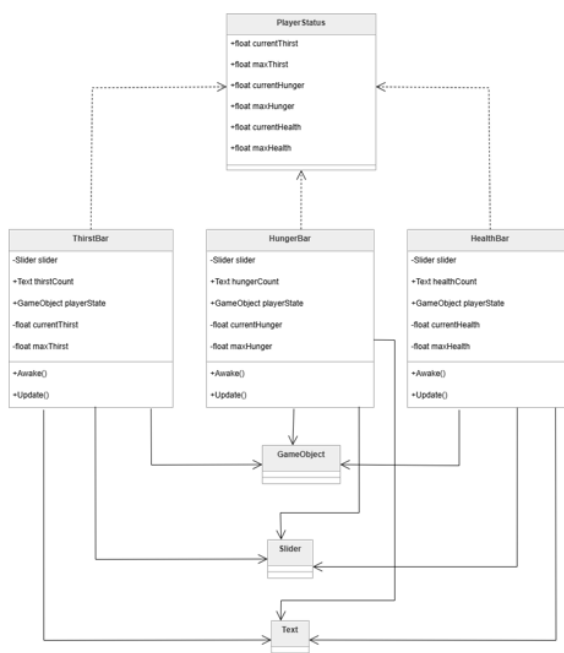
9

ДІАГРАМА КЛАСІВ СИСТЕМИ МЕНЮ



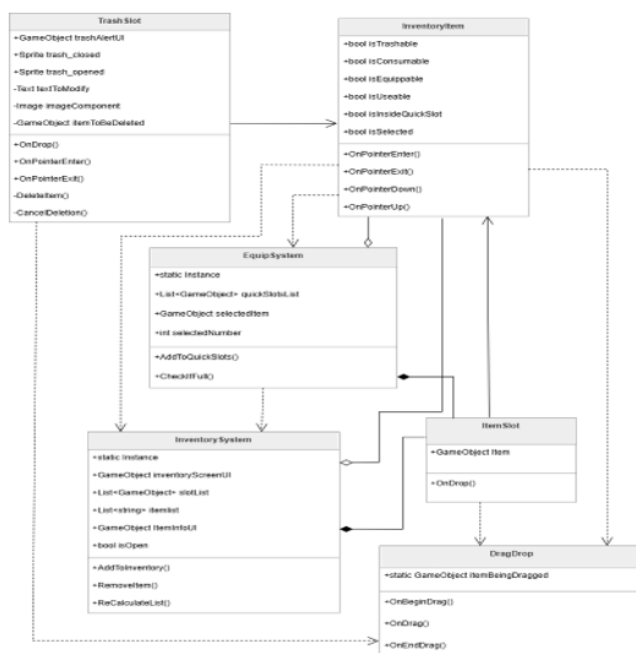
10

ДІАГРАМА КЛАСІВ СИСТЕМИ ВІДОБРАЖЕННЯ СТАНУ ПЕРСОНАЖА



11

ДІАГРАМА КЛАСІВ СИСТЕМИ ІНВЕНТАРЮ



12

ЕКРАНІ ФОРМИ: ГОЛОВНЕ МЕНЮ

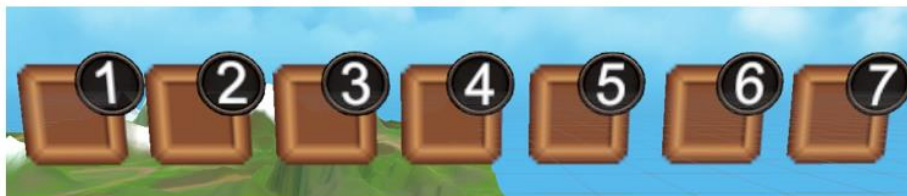


13

ЕКРАНІ ФОРМИ: МЕНЮ ПАУЗИ



14

ЕКРАНІ ФОРМИ: ПОТВАР ТА ПОКАЗНИКИ ГРАВЦЯ

15

ЕКРАНІ ФОРМИ: ІНВЕНТАР ГРАВЦЯ

16

ЕКРАНІ ФОРМИ: ВИГЛЯД МАПИ ЗГОРИ



17

ЕКРАНІ ФОРМИ: ВИГЛЯД МАПИ З ПОГЛЯДУ ГРАВЦЯ



18

ЕКРАНІ ФОРМИ: МІСТЕЧНО З НРС

19

ЕКРАНІ ФОРМИ: НРС КОЙОТ

20

ЕКРАНІ ФОРМИ: ФЕРМА



21

ЕКРАНІ ФОРМИ: ЦВИНТАР



22

ЕКРАНІ ФОРМИ: ЗАКИНУТИЙ ТАБІР



23

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Ромашко В.В., Замрій І.В. Архітектура 3D гри в жанрі survival з елементами RPG. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, ДУІКТ. Збірник тез. – К.: ДУІКТ, 2025. С.468 -471.
2. Ромашко В.В., Замрій І.В. Визначення вимог до Frontend-частини 3D гри в жанрі survival з елементами RPG. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційнокомунікаційних технологіях», 23 квітня 2026 року, Київ, ДУІКТ. Збірник тез. – К.: ДУІКТ, 2026. С.128 -130.

24

ВИСНОВКИ

1. Проведений аналіз відеоігор жанру виживання з елементами RPG дозволив визначити ключові механіки, особливості frontend-частини та цільову аудиторію цього напрямку.
2. Проведений порівняльний аналіз frontend-реалізації у популярних іграх-аналогах дозволив не лише виявити найбільш ефективні підходи до побудови користувацького інтерфейсу, а й на основі отриманих результатів сформулювати функціональні та нефункціональні вимоги.
3. У процесі дослідження обрано допільні засоби реалізації frontend-частини відеогри з урахуванням сучасних технологій та вимог проєкту, зокрема ігровий рушій Unity, мову програмування C# та середовище розробки Visual Studio.
4. На основі проведеного аналізу визначено візуальний дизайн та стиль гри, що відповідає обраній тематиці, а також спроєктовано ігровий світ, який охоплює містечко з NPC, ферму, цвинтар і закинтий табір, та успішно реалізовано ключові елементи інтерфейсу, зокрема головне меню, меню паузи, систему інвентарю та візуалізацію поточних станів персонажа.
5. Після завершення основних етапів розробки проведено тестування створених механік та ігрової мапи, що дозволило оцінити їх працездатність і виявити можливі недоліки.

25

ДЯКУЮ ЗА УВАГУ!

26

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

Вихідний код файлу InventorySystem

```

using System;
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.UI;

public class InventorySystem : MonoBehaviour
{
    public static InventorySystem Instance { get; set; }

    public GameObject inventoryScreenUI;
    public List<GameObject> slotList = new List<GameObject>();
    public List<string> itemList = new List<string>();

    private GameObject itemToAdd;
    private GameObject whatSlotToEquip;
    public GameObject ItemInfoUI;

    public bool isOpen;
    //public bool isFull;

    //pickupsPop
    public GameObject pickupAlert;
    public Text pickupName;
    public Image pickupImage;

    public List<string> itemsPicked;

    private void Awake()
    {
        if (Instance != null && Instance != this)
        {
            Destroy(gameObject);
        }
        else
        {
            Instance = this;
        }

        void Start()
        {
            isOpen = false;
            PopulateSlotList();
            Cursor.visible = false;
        }

        private void PopulateSlotList()
        {
            foreach (Transform child in inventoryScreenUI.transform)
            {
                if (child.CompareTag("Slot"))
                {
                    slotList.Add(child.gameObject);
                }
            }
        }

        void Update()
        {
            if (Input.GetKeyDown(KeyCode.I) && !isOpen )
        
```

```

{

    inventoryScreenUI.SetActive(true);

    Cursor.lockState = CursorLockMode.None;

    Cursor.visible = true;

    SelectionManager.instance.DisableSelection();

SelectionManager.instance.GetComponent<SelectionManager>
>().enabled = false;

    isOpen = true;

}

else if (Input.GetKeyDown(KeyCode.I) && isOpen)
{
    inventoryScreenUI.SetActive(false);

    if (!CraftingSystem.instance.isOpen)
    {
        Cursor.lockState = CursorLockMode.Locked;

        Cursor.visible = false;

        SelectionManager.instance.EnableSelection();

SelectionManager.instance.GetComponent<SelectionManager>
>().enabled = true;

        }

        isOpen = false;

    }

}

public void AddToInventory(string itemName)
{
    whatSlotToEquip = FindNextEmptySlot();

    itemToAdd =
(GameObject)Instantiate(Resources.Load<GameObject>(item
Name), whatSlotToEquip.transform.position,
whatSlotToEquip.transform.rotation);

    itemToAdd.transform.SetParent(whatSlotToEquip.transform);

    itemlist.Add(itemName);

    TriggerPickUpPop(itemName,
itemToAdd.GetComponent<Image>().sprite);

    ReCalculateList();

    CraftingSystem.instance.RefreshNeededItems();

}

void TriggerPickUpPop(string itemName, Sprite itemSprite)
{
    pickupAlert.SetActive(true);

    pickupName.text = itemName;

    pickupImage.sprite = itemSprite;

    StartCoroutine(HidePopupAfterDelay(3f));

}

IEnumerator HidePopupAfterDelay(float delay)
{
    yield return new WaitForSeconds(delay);

    pickupAlert.SetActive(false);

}

private GameObject FindNextEmptySlot()
{
    foreach(GameObject slot in slotList)
    {
        if(slot.transform.childCount == 0)
        {
            return slot;

        }

    }

    return new GameObject();

}

public bool CheckifFull()
{
    int counter = 0;

    foreach (GameObject slot in slotList)

```

```

{
    if (slot.transform.childCount > 0)
    {
        counter += 1;
    }
}

if (counter == 28)
{
    return true;
}
else
{
    return false;
}
}

public void RemoveItem(string nameToRemove, int
amountToRemove)
{
    int counter = amountToRemove;
    for(var i = slotList.Count - 1; i >= 0; i--)
    {
        if (slotList[i].transform.childCount > 0)
        {
            if (slotList[i].transform.GetChild(0).name ==
nameToRemove + "(Clone)" && counter != 0)
                DestroyImmediate(slotList[i].transform.GetChild(0).gameObje
ct);
            counter--;
        }
    }
    ReCalculateList();
    CraftingSystem.instance.RefreshNeededItems();
}

public void ReCalculateList()
{
    itemlist.Clear();
    foreach (GameObject slot in slotList)
    {
        if (slot.transform.childCount > 0)
        {
            string name = slot.transform.GetChild(0).name;
            string str2 = "(Clone)";
            string result = name.Replace(str2, "");
            itemlist.Add(result);
        }
    }
}
}

```

Вихідний код файлу MainMenuController

```

using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class MainMenuController : MonoBehaviour
{
    public Button LoadGamebtn;

    public void StartNewGame()
    {
        SceneManager.LoadScene("GameScene");
    }

    public void ExitGame()
    {
        Debug.Log("Quit");
        Application.Quit();
    }
}

```

Вихідний код файлу MenuManager

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MenuManager : MonoBehaviour
{
    public static MenuManager Instance { get; set; }
    public GameObject menuCanvas;
    public GameObject UICanvas;

    public GameObject saveMenu;
    public GameObject settingMenu;
    public GameObject menu;

    public bool isMenuOPEn;

    private void Awake()
    {
        if (Instance != null && Instance != this)
        {
            Destroy(gameObject);
        }
        else
        {
            Instance = this;
        }
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.M) && !isMenuOPEn)
        {
            UICanvas.SetActive(false);

            menuCanvas.SetActive(true);
            isMenuOPEn = true;

            Cursor.lockState = CursorLockMode.None;
            Cursor.visible = true;

            SelectionManager.instance.DisableSelection();

            SelectionManager.instance.GetComponent<SelectionManager>().enabled = false;

            }else if(Input.GetKeyDown(KeyCode.M) && isMenuOPEn)
            {
                saveMenu.SetActive(false);
                settingMenu.SetActive(false);
                menu.SetActive(true);
                UICanvas.SetActive(true);
                menuCanvas.SetActive(false);
                isMenuOPEn = false;

                if (CraftingSystem.instance.isOpen == false && InventorySystem.Instance.isOpen == false)
                {
                    Cursor.lockState = CursorLockMode.Locked;
                    Cursor.visible = false;
                }

                SelectionManager.instance.EnableSelection();

                SelectionManager.instance.GetComponent<SelectionManager>().enabled = true;
            }
        }
    }
}

```

Вихідний код файлу HealthBar

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class HealthBar : MonoBehaviour
{
    private Slider slider;
    public Text healthCount;

    public GameObject playerState;
    private float currentHealth, maxHealth;
    // Start is called before the first frame update
    void Awake()
    {

```

```

        slider = GetComponent<Slider>();
    }

    // Update is called once per frame
    void Update()
    {
        currentHealth = playerState.GetComponent <
        PlayerStatus>().currentHealth;

        maxHealth =
        playerState.GetComponent<PlayerStatus>().maxHealth;

        float fillValue = currentHealth / maxHealth;
        slider.value = fillValue;
        healthCount.text = currentHealth + "/" + maxHealth;
    }
}

```

Вихідний код файлу HungerBar

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class HungerBar : MonoBehaviour
{
    private Slider slider;
    public Text hungerCount;

    public GameObject playerState;
    private float currentHunger, maxHunger;
    // Start is called before the first frame update
    void Awake()
    {

```

```

        slider = GetComponent<Slider>();
    }

    // Update is called once per frame
    void Update()
    {
        currentHunger =
        playerState.GetComponent<PlayerStatus>().currentHunger;

        maxHunger =
        playerState.GetComponent<PlayerStatus>().maxHunger;

        float fillValue = currentHunger / maxHunger;
        slider.value = fillValue;
        hungerCount.text = currentHunger + "/" + maxHunger;
    }
}

```

Вихідний код файлу ThirstBar

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

```

```

public class ThirstBar : MonoBehaviour
{
    private Slider slider;

```

```
public Text thirstCount;

public GameObject playerState;
private float currentThirst, maxThirst;
// Start is called before the first frame update
void Awake()
{
    slider = GetComponent<Slider>();
}

// Update is called once per frame
```

```
void Update()
{
    currentThirst =
playerState.GetComponent<PlayerStatus>().currentThirst;

    maxThirst =
playerState.GetComponent<PlayerStatus>().maxThirst;

    float fillValue = currentThirst / maxThirst;
    slider.value = fillValue;

    thirstCount.text = currentThirst + "%" ;
}
}
```